```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import warnings
warnings.filterwarnings('ignore')

# Set style for better plots
plt.style.use('default')
sns.set_palette("husl")

print("=" * 80)
print("DATA ANALYSIS ASSIGNMENTS - WORKING WITH CSV FILES")
print("=" * 80)

# First, let's create the dummy CSV files that you can download and use
print("\n🔥 CREATING DUMMY CSV FILES FOR DOWNLOAD")
print("-" * 50)

#
================================================================================
===
# CREATE CSV FILE 1: SCOTTISH HILLS DATA
#
================================================================================
===
```

```
import numpy as np
np.random.seed(42)

# Generate Scottish Hills dataset
n_hills = 150
hills_data = {
   'Hill_Name': [f'Ben_{chr(65+i//10)}{i%10+1}' for i in range(n_hills)],
   'Height_m': np.random.normal(800, 250, n_hills),
   'Latitude': np.random.uniform(55.8, 58.5, n_hills),
   'Longitude': np.random.uniform(-5.5, -2.8, n_hills),
   'Distance_km': np.random.exponential(8, n_hills),
   'Time_hours': np.random.gamma(3, 1.5, n_hills),
   'Difficulty': np.random.choice(['Easy', 'Moderate', 'Hard', 'Extreme'], n_hills, p=[0.2, 0.4, 0.3, 0.1]),
   'Region': np.random.choice(['Highlands', 'Lowlands', 'Cairngorms', 'Grampians'], n_hills, p=[0.4, 0.2, 0.25, 0.15])
}

# Ensure positive values
hills_data['Height_m'] = np.abs(hills_data['Height_m'])
hills_data['Distance_km'] = np.abs(hills_data['Distance_km'])
hills_data['Time_hours'] = np.abs(hills_data['Time_hours'])

# Add some missing values to simulate real data
hills_df = pd.DataFrame(hills_data)
hills_df.loc[np.random.choice(n_hills, 15, replace=False), 'Time_hours'] = np.nan
hills_df.loc[np.random.choice(n_hills, 8, replace=False), 'Distance_km'] = np.nan
```

```
# Save CSV file
hills_df.to_csv('scottish_hills_data.csv', index=False)
print("✅ Created: scottish_hills_data.csv")


#
================================================================================
# CREATE CSV FILE 2: TITANIC DATA
#
================================================================================
np.random.seed(123)


n_passengers = 891
titanic_data = {
  'PassengerId': range(1, n_passengers + 1),
  'Survived': np.random.choice([0, 1], n_passengers, p=[0.616, 0.384]),
  'Pclass': np.random.choice([1, 2, 3], n_passengers, p=[0.242, 0.206, 0.552]),
  'Name': [f'Passenger_{i}' for i in range(1, n_passengers + 1)],
  'Sex': np.random.choice(['male', 'female'], n_passengers, p=[0.647, 0.353]),
  'Age': np.random.normal(29.7, 14.5, n_passengers),
  'SibSp': np.random.choice([0, 1, 2, 3, 4, 5], n_passengers, p=[0.68, 0.23, 0.06, 0.02,
0.008, 0.002]),
  'Parch': np.random.choice([0, 1, 2, 3, 4, 5, 6], n_passengers,
                p=[0.76, 0.13, 0.08, 0.005, 0.004, 0.001, 0.02]),
  'Ticket': [f'TICKET_{i}' for i in range(1, n_passengers + 1)],
```

```
    'Fare': np.random.lognormal(2.5, 1.2, n_passengers),
    'Cabin': [f'C{i}' if np.random.random() > 0.77 else np.nan for i in range(n_passengers)],
    # 'Embarked': np.random.choice(['C', 'Q', 'S'], n_passengers, p=[0.188, 0.086, 0.724])
}

# Ensure realistic age values
titanic_data['Age'] = np.clip(titanic_data['Age'], 0.42, 80)
# Add missing ages
titanic_df = pd.DataFrame(titanic_data)
titanic_df.loc[np.random.choice(n_passengers, 177, replace=False), 'Age'] = np.nan

# Create class names
titanic_df['Class'] = titanic_df['Pclass'].map({1: 'First', 2: 'Second', 3: 'Third'})

titanic_df.to_csv('titanic_dataset.csv', index=False)
print("✅ Created: titanic_dataset.csv")

#
================================================================================
===
# CREATE CSV FILE 3: STUDENT PERFORMANCE DATA
#
================================================================================
===
np.random.seed(456)

n_students = 500
```

```
majors = ["Computer Science", "Mathematics", "Physics", "Biology", "Chemistry", "Engineering", "Economics", "Psychology"]

departments = ["STEM", "Liberal Arts", "Business", "Social Sciences"]


students_data = {
    'Student_ID': [f'STU{1000 + i}' for i in range(n_students)],
    'Name': [f'Student_{chr(65+i//26)}{chr(65+i%26)}' for i in range(n_students)],
    'Age': np.random.choice(range(18, 26), n_students),
    'Gender': np.random.choice(['Male', 'Female', 'Other'], n_students, p=[0.48, 0.50, 0.02]),
    'Major': np.random.choice(majors, n_students),
    'Year': np.random.choice([1, 2, 3, 4], n_students, p=[0.25, 0.25, 0.25, 0.25]),
    'GPA': np.random.beta(7, 2, n_students) * 2 + 2,  # Creates realistic GPA distribution
    'Credits_Completed': np.random.randint(12, 150, n_students),
    'Study_Hours_Week': np.random.gamma(3, 5, n_students),
    'Extracurricular_Hours': np.random.exponential(3, n_students),
    'Part_Time_Job': np.random.choice(['Yes', 'No'], n_students, p=[0.6, 0.4]),
    'Scholarship': np.random.choice(['Yes', 'No'], n_students, p=[0.3, 0.7]),
    'Hometown_Distance_km': np.random.lognormal(4, 1.5, n_students)
}

# Ensure realistic values
students_data['GPA'] = np.clip(students_data['GPA'], 2.0, 4.0)
students_data['Study_Hours_Week'] = np.clip(students_data['Study_Hours_Week'], 5, 60)
students_data['Extracurricular_Hours'] = np.clip(students_data['Extracurricular_Hours'], 0, 25)


students_df = pd.DataFrame(students_data)
```

```
students_df.to_csv('student_performance_data.csv', index=False)
print("✅ Created: student_performance_data.csv")

print("\n📁 CSV FILES CREATED - DOWNLOAD LINKS:")
print("-" * 50)
print("1. scottish_hills_data.csv - For Assignment 1")
print("2. titanic_dataset.csv - For Assignment 2")
print("3. student_performance_data.csv - For Assignment 3")
print("\n💡 Note: These files are created in your current working directory")
print("You can download them and use with the code below.")

print("\n" + "=" * 80)
print("ASSIGNMENT 1: EXPLORATORY DATA ANALYSIS WITH CSV FILE")
print("=" * 80)

#
==========================================================================
===
# ASSIGNMENT 1: SCOTTISH HILLS DATA ANALYSIS
#
==========================================================================
===

print("\n📂 LOADING DATA FROM CSV FILE")
print("-" * 40)

# Load the CSV file
```

```
df_hills = pd.read_csv('scottish_hills_data.csv')

print("✅ Successfully loaded scottish_hills_data.csv")

print(f"Dataset shape: {df_hills.shape}")


print("\n1️⃣ NUMPY OPERATIONS:")

print("-" * 30)


# Using NumPy: Load numerical column as array

heights_array = df_hills['Height_m'].values

print(f"Loaded {len(heights_array)} height values into NumPy array")


# Compute statistics

mean_height = np.mean(heights_array)

variance_height = np.var(heights_array)

std_height = np.std(heights_array)


print(f"Mean height: {mean_height:.2f} m")

print(f"Variance: {variance_height:.2f}")

print(f"Standard deviation: {std_height:.2f} m")


# Filter values above and below mean

above_mean = heights_array[heights_array > mean_height]

below_mean = heights_array[heights_array < mean_height]


print(f"Hills above mean height: {len(above_mean)}
({len(above_mean)/len(heights_array)*100:.1f}%)")
```

```
print(f"Hills below mean height: {len(below_mean)}
({len(below_mean)/len(heights_array)*100:.1f}%)")


print(f"\nHeights above mean (first 10): {above_mean[:10]}")

print(f"Heights below mean (first 10): {below_mean[:10]}")


print("\n 2️⃣ PANDAS OPERATIONS:")

print("-" * 30)


print("Dataset head:")

print(df_hills.head())


print(f"\nDataset info:")

df_hills.info()


print(f"\nDataset description:")

print(df_hills.describe())


# Handle missing values

print(f"\nMissing values before handling:")

print(df_hills.isnull().sum())


# Fill missing values with appropriate methods

df_hills['Time_hours'].fillna(df_hills['Time_hours'].median(), inplace=True)

df_hills['Distance_km'].fillna(df_hills['Distance_km'].mean(), inplace=True)
```

```
print(f"\nMissing values after handling:")
print(df_hills.isnull().sum())

# Create new column based on existing data
def categorize_height(height):
    if height > mean_height + std_height:
        return 'High'
    elif height < mean_height - std_height:
        return 'Low'
    else:
        return 'Medium'

df_hills['Height_Category'] = df_hills['Height_m'].apply(categorize_height)
print(f"\nHeight categories distribution:")
print(df_hills['Height_Category'].value_counts())

# Additional analysis
print(f"\nAnalysis by Region:")
region_stats = df_hills.groupby('Region')['Height_m'].agg(['mean', 'std', 'count'])
print(region_stats)

print("\n 3️⃣ MATPLOTLIB VISUALIZATION:")
print("-" * 30)

# Create comprehensive visualizations
plt.figure(figsize=(16, 12))
```

```
# 1. Scatter plot with linear regression
plt.subplot(2, 3, 1)
plt.scatter(df_hills['Height_m'], df_hills['Latitude'], alpha=0.6, color='blue')

# Add linear regression line
slope, intercept, r_value, p_value, std_err = stats.linregress(df_hills['Height_m'],
df_hills['Latitude'])
line = slope * df_hills['Height_m'] + intercept
plt.plot(df_hills['Height_m'], line, 'r--', linewidth=2,
    label=f'y = {slope:.4f}x + {intercept:.2f}\nR^2 = {r_value**2:.3f}')

plt.xlabel('Height (m)')
plt.ylabel('Latitude')
plt.title('Height vs Latitude of Scottish Hills')
plt.legend()
plt.grid(True, alpha=0.3)

# 2. Height distribution histogram
plt.subplot(2, 3, 2)
plt.hist(df_hills['Height_m'], bins=25, alpha=0.7, color='green', edgecolor='black')
plt.axvline(mean_height, color='red', linestyle='--', linewidth=2, label=f'Mean:
{mean_height:.1f}m')
plt.axvline(mean_height + std_height, color='orange', linestyle=':', label=f'Mean + 1σ:
{mean_height + std_height:.1f}m')
plt.axvline(mean_height - std_height, color='orange', linestyle=':', label=f'Mean - 1σ:
{mean_height - std_height:.1f}m')
```

```
plt.xlabel('Height (m)')
plt.ylabel('Frequency')
plt.title('Distribution of Hill Heights')
plt.legend()
plt.grid(True, alpha=0.3)

# 3. Height categories bar chart
plt.subplot(2, 3, 3)
height_cats = df_hills['Height_Category'].value_counts()
bars = plt.bar(height_cats.index, height_cats.values, color=['red', 'orange', 'green'],
alpha=0.7)
for bar, value in zip(bars, height_cats.values):
   plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.5,
         str(value), ha='center', va='bottom', fontweight='bold')
plt.xlabel('Height Category')
plt.ylabel('Count')
plt.title('Distribution of Height Categories')
plt.grid(True, alpha=0.3)

# 4. Distance vs Time scatter plot
plt.subplot(2, 3, 4)
scatter = plt.scatter(df_hills['Distance_km'], df_hills['Time_hours'],
               c=df_hills['Height_m'], cmap='viridis', alpha=0.7, s=60)
plt.colorbar(scatter, label='Height (m)')
plt.xlabel('Distance (km)')
plt.ylabel('Time (hours)')
```

```
plt.title('Distance vs Time (colored by height)')
plt.grid(True, alpha=0.3)

# 5. Box plot by region
plt.subplot(2, 3, 5)
regions = df_hills['Region'].unique()
region_heights = [df_hills[df_hills['Region'] == region]['Height_m'].values for region in regions]
box_plot = plt.boxplot(region_heights, labels=regions, patch_artist=True)
colors = ['lightblue', 'lightgreen', 'lightcoral', 'lightyellow']
for patch, color in zip(box_plot['boxes'], colors):
    patch.set_facecolor(color)
plt.xlabel('Region')
plt.ylabel('Height (m)')
plt.title('Height Distribution by Region')
plt.xticks(rotation=45)
plt.grid(True, alpha=0.3)

# 6. Difficulty vs Height
plt.subplot(2, 3, 6)
difficulty_order = ['Easy', 'Moderate', 'Hard', 'Extreme']
df_hills['Difficulty'] = pd.Categorical(df_hills['Difficulty'], categories=difficulty_order, ordered=True)
avg_height_by_difficulty = df_hills.groupby('Difficulty')['Height_m'].mean()
bars = plt.bar(avg_height_by_difficulty.index, avg_height_by_difficulty.values,
               color=['green', 'yellow', 'orange', 'red'], alpha=0.7)
for bar, value in zip(bars, avg_height_by_difficulty.values):
```

```
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 10,
             f'{value:.0f}m', ha='center', va='bottom', fontweight='bold')
plt.xlabel('Difficulty Level')
plt.ylabel('Average Height (m)')
plt.title('Average Height by Difficulty Level')
plt.xticks(rotation=45)
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('assignment1_scottish_hills_analysis.png', dpi=300, bbox_inches='tight')
plt.show()

print("✅ Assignment 1 plots saved as 'assignment1_scottish_hills_analysis.png'")

print("\n" + "=" * 80)
print("ASSIGNMENT 2: ADVANCED VISUALIZATION WITH TITANIC CSV")
print("=" * 80)

#
================================================================================
===
# ASSIGNMENT 2: TITANIC DATA ANALYSIS
#
================================================================================
===

print("\n📂 LOADING TITANIC DATA FROM CSV FILE")
```

```
print("-" * 40)

# Load the Titanic CSV file
df_titanic = pd.read_csv('titanic_dataset.csv')
print(" ✅ Successfully loaded titanic_dataset.csv")
print(f"Dataset shape: {df_titanic.shape}")

print("\n 1️⃣ PANDAS OPERATIONS:")
print("-" * 30)

print("Dataset head:")
print(df_titanic.head())

print(f"\nDataset info:")
df_titanic.info()

print(f"\nBasic statistics:")
print(df_titanic.describe())

# Basic cleaning
print(f"\nMissing values before cleaning:")
print(df_titanic.isnull().sum())

# Fill missing ages with median by class and sex
df_titanic['Age'] = df_titanic.groupby(['Pclass', 'Sex'])['Age'].transform(
    lambda x: x.fillna(x.median()))
```

```
# Fill remaining missing ages with overall median
df_titanic['Age'].fillna(df_titanic['Age'].median(), inplace=True)

# Fill missing embarked with mode
df_titanic['Embarked'].fillna(df_titanic['Embarked'].mode()[0], inplace=True)

print(f"\nMissing values after cleaning:")
print(df_titanic.isnull().sum())

# Group by class and calculate average age
avg_age_by_class = df_titanic.groupby('Class')['Age'].mean()
print(f"\nAverage age by class:")
for class_name, avg_age in avg_age_by_class.items():
    print(f"  {class_name}: {avg_age:.1f} years")

# Additional analysis
survival_stats = df_titanic.groupby(['Class', 'Sex'])['Survived'].agg(['count', 'sum', 'mean'])
print(f"\nSurvival statistics by class and gender:")
print(survival_stats)

print("\n 2️⃣ PANDAS BUILT-IN PLOTTING:")
print("-" * 30)

plt.figure(figsize=(18, 12))
```

```
# Pandas boxplot
plt.subplot(2, 4, 1)
df_titanic.boxplot(column='Age', by='Class', ax=plt.gca())
plt.title('Age Distribution by Class (Pandas)')
plt.xlabel('Class')
plt.ylabel('Age')
plt.suptitle('')

print("\n 3️⃣ SEABORN VISUALIZATIONS:")
print("-" * 30)

# 1. Seaborn boxplot
plt.subplot(2, 4, 2)
sns.boxplot(data=df_titanic, x='Class', y='Age', palette='Set2')
plt.title('Age Distribution by Class (Seaborn)')
plt.xlabel('Class')
plt.ylabel('Age')

# 2. Scatter plot of fare vs age, colored by survival
plt.subplot(2, 4, 3)
sns.scatterplot(data=df_titanic, x='Age', y='Fare', hue='Survived', alpha=0.6)
plt.title('Fare vs Age (colored by survival)')
plt.xlabel('Age')
plt.ylabel('Fare')
plt.yscale('log')  # Log scale for better visualization of fare
```

```
# 3. Enhanced scatter plot with multiple variables
plt.subplot(2, 4, 4)
sns.scatterplot(data=df_titanic, x='Age', y='Fare', hue='Survived',
                size='Pclass', style='Sex', alpha=0.7)
plt.title('Fare vs Age (multi-variable)')
plt.xlabel('Age')
plt.ylabel('Fare')
plt.yscale('log')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')

# 4. Survival rate by class
plt.subplot(2, 4, 5)
survival_by_class = df_titanic.groupby('Class')['Survived'].mean()
bars = plt.bar(survival_by_class.index, survival_by_class.values,
               color=['gold', 'silver', '#CD7F32'], alpha=0.8)
for bar, value in zip(bars, survival_by_class.values):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.01,
             f'{value:.1%}', ha='center', va='bottom', fontweight='bold')
plt.title('Survival Rate by Class')
plt.xlabel('Class')
plt.ylabel('Survival Rate')
plt.ylim(0, 1)
plt.grid(True, alpha=0.3)

# 5. Count plot of survival by class
plt.subplot(2, 4, 6)
```

```
sns.countplot(data=df_titanic, x='Class', hue='Survived', palette='Set1')
plt.title('Survival Count by Class')
plt.xlabel('Class')
plt.ylabel('Count')

# 6. Survival by gender and class
plt.subplot(2, 4, 7)
survival_heatmap = df_titanic.pivot_table(values='Survived', index='Sex', columns='Class', aggfunc='mean')
sns.heatmap(survival_heatmap, annot=True, fmt='.2f', cmap='RdYlGn',
        cbar_kws={'label': 'Survival Rate'})
plt.title('Survival Rate Heatmap')
plt.xlabel('Class')
plt.ylabel('Gender')

# 7. Age distribution by survival
plt.subplot(2, 4, 8)
plt.hist(df_titanic[df_titanic['Survived'] == 0]['Age'], bins=30, alpha=0.5,
      label='Did not survive', color='red', density=True)
plt.hist(df_titanic[df_titanic['Survived'] == 1]['Age'], bins=30, alpha=0.5,
      label='Survived', color='green', density=True)
plt.xlabel('Age')
plt.ylabel('Density')
plt.title('Age Distribution by Survival')
plt.legend()
plt.grid(True, alpha=0.3)
```

```
plt.tight_layout()
plt.savefig('assignment2_titanic_analysis.png', dpi=300, bbox_inches='tight')
plt.show()

print("✅ Assignment 2 plots saved as 'assignment2_titanic_analysis.png'")

print("\n" + "=" * 80)
print("ASSIGNMENT 3: STUDENT PERFORMANCE DATA FROM CSV")
print("=" * 80)

#
================================================================================
===
# ASSIGNMENT 3: STUDENT PERFORMANCE ANALYSIS
#
================================================================================
===

print("\n📂 LOADING STUDENT DATA FROM CSV FILE")
print("-" * 40)

# Load the student performance CSV file
df_students = pd.read_csv('student_performance_data.csv')
print("✅ Successfully loaded student_performance_data.csv")
print(f"Dataset shape: {df_students.shape}")
```

```
print("\nPART 1: DATA OVERVIEW")
print("-" * 25)

print("Dataset head:")
print(df_students.head())

print(f"\nDataset info:")
df_students.info()

print(f"\nDataset description:")
print(df_students.describe())

print("\nPART 2: DATA PROCESSING WITH PANDAS & NUMPY")
print("-" * 45)

# Check for duplicates and missing values
print(f"Duplicate records: {df_students.duplicated().sum()}")
print(f"Missing values:")
print(df_students.isnull().sum())

# Remove any duplicates
df_students = df_students.drop_duplicates()
print(f"Final dataset shape after cleaning: {df_students.shape}")

# NumPy operations on GPA
gpa_array = df_students['GPA'].values
```

```
mean_gpa = np.mean(gpa_array)

std_gpa = np.std(gpa_array)

median_gpa = np.median(gpa_array)

min_gpa = np.min(gpa_array)

max_gpa = np.max(gpa_array)


print(f"\n 📊 GPA STATISTICS (using NumPy):")

print(f"  Mean GPA: {mean_gpa:.3f}")

print(f"  Standard deviation: {std_gpa:.3f}")

print(f"  Median GPA: {median_gpa:.3f}")

print(f"  Range: {min_gpa:.3f} - {max_gpa:.3f}")


# Filter students above mean GPA

above_mean_mask = gpa_array > mean_gpa

above_mean_students = df_students[above_mean_mask]

print(f"\n 🎯 Students with GPA above mean ({mean_gpa:.3f}): {len(above_mean_students)}
({len(above_mean_students)/len(df_students)*100:.1f}%)")


print("\nTop 10 students with highest GPA:")

top_students = df_students.nlargest(10, 'GPA')[['Student_ID', 'Name', 'GPA', 'Major',
'Study_Hours_Week']]

print(top_students)


# Add Performance column

def categorize_performance(gpa):

    if gpa >= 3.5:
```

```
        return 'Excellent'
    elif gpa >= 3.0:
        return 'Good'
    else:
        return 'Needs Improvement'

df_students['Performance'] = df_students['GPA'].apply(categorize_performance)

print(f"\n📈 PERFORMANCE DISTRIBUTION:")
perf_counts = df_students['Performance'].value_counts()
for perf, count in perf_counts.items():
    print(f"  {perf}: {count} students ({count/len(df_students)*100:.1f}%)")

print("\nPART 3: EXPLORATORY DATA ANALYSIS & VISUALIZATION")
print("-" * 50)

# Create comprehensive visualizations
plt.figure(figsize=(20, 15))

# 1. Histogram of GPAs
plt.subplot(3, 4, 1)
plt.hist(df_students['GPA'], bins=30, alpha=0.7, color='skyblue', edgecolor='black',
density=True)
plt.axvline(mean_gpa, color='red', linestyle='--', linewidth=2, label=f'Mean: {mean_gpa:.2f}')
plt.axvline(median_gpa, color='green', linestyle='--', linewidth=2, label=f'Median:
{median_gpa:.2f}')
```

```
plt.xlabel('GPA')
plt.ylabel('Density')
plt.title('Distribution of Student GPAs')
plt.legend()
plt.grid(True, alpha=0.3)

# 2. Bar chart of average GPA by major
plt.subplot(3, 4, 2)
avg_gpa_by_major =
df_students.groupby('Major')['GPA'].mean().sort_values(ascending=False)
bars = plt.bar(range(len(avg_gpa_by_major)), avg_gpa_by_major.values,
          color=plt.cm.Set3(np.linspace(0, 1, len(avg_gpa_by_major))))
plt.xlabel('Major')
plt.ylabel('Average GPA')
plt.title('Average GPA by Major')
plt.xticks(range(len(avg_gpa_by_major)), avg_gpa_by_major.index, rotation=45, ha='right')
plt.grid(True, alpha=0.3)

# Add value labels
for i, (bar, value) in enumerate(zip(bars, avg_gpa_by_major.values)):
   plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.01,
         f'{value:.2f}', ha='center', va='bottom', fontsize=8)

# 3. Boxplot of GPA by Performance
plt.subplot(3, 4, 3)
performance_order = ['Needs Improvement', 'Good', 'Excellent']
```

```
sns.boxplot(data=df_students, x='Performance', y='GPA', order=performance_order,
            palette=['red', 'orange', 'green'])
plt.title('GPA Distribution by Performance Category')
plt.xticks(rotation=45)

# 4. Scatter plot of Age vs GPA colored by Major
plt.subplot(3, 4, 4)
majors = df_students['Major'].unique()
colors = plt.cm.tab10(np.linspace(0, 1, len(majors)))

for major, color in zip(majors, colors):
    major_data = df_students[df_students['Major'] == major]
    plt.scatter(major_data['Age'], major_data['GPA'],
                label=major, alpha=0.7, color=color, s=30)

plt.xlabel('Age')
plt.ylabel('GPA')
plt.title('Age vs GPA by Major')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', fontsize=8)
plt.grid(True, alpha=0.3)

# 5. Study hours vs GPA
plt.subplot(3, 4, 5)
plt.scatter(df_students['Study_Hours_Week'], df_students['GPA'],
            c=df_students['Year'], cmap='viridis', alpha=0.6, s=40)
plt.colorbar(label='Academic Year')
```

```
plt.xlabel('Study Hours per Week')
plt.ylabel('GPA')
plt.title('Study Hours vs GPA (colored by year)')
plt.grid(True, alpha=0.3)

# Add correlation coefficient
corr_coef = np.corrcoef(df_students['Study_Hours_Week'], df_students['GPA'])[0, 1]
plt.text(0.05, 0.95, f'r = {corr_coef:.3f}', transform=plt.gca().transAxes,
        bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

# 6. Performance distribution pie chart
plt.subplot(3, 4, 6)
performance_counts = df_students['Performance'].value_counts()
colors_pie = ['red', 'orange', 'green']
wedges, texts, autotexts = plt.pie(performance_counts.values,
labels=performance_counts.index,
                                   autopct='%1.1f%%', colors=colors_pie, startangle=90)
plt.title('Distribution of Student Performance')

# 7. Major distribution
plt.subplot(3, 4, 7)
major_counts = df_students['Major'].value_counts()
bars = plt.bar(range(len(major_counts)), major_counts.values,
             color=plt.cm.Set2(np.linspace(0, 1, len(major_counts))))
plt.xlabel('Major')
plt.ylabel('Number of Students')
```

```
plt.title('Distribution of Students by Major')
plt.xticks(range(len(major_counts)), major_counts.index, rotation=45, ha='right')
plt.grid(True, alpha=0.3)

# 8. GPA vs Credits completed
plt.subplot(3, 4, 8)
plt.scatter(df_students['Credits_Completed'], df_students['GPA'],
        c=df_students['Year'], cmap='plasma', alpha=0.6, s=40)
plt.colorbar(label='Academic Year')
plt.xlabel('Credits Completed')
plt.ylabel('GPA')
plt.title('Credits vs GPA (colored by year)')
plt.grid(True, alpha=0.3)

# 9. Extracurricular activities vs GPA
plt.subplot(3, 4, 9)
# Create bins for extracurricular hours
df_students['Extra_Bins'] = pd.cut(df_students['Extracurricular_Hours'],
                    bins=[0, 5, 10, 15, 25],
                    labels=['0-5h', '5-10h', '10-15h', '15+ h'])
sns.boxplot(data=df_students, x='Extra_Bins', y='GPA', palette='Set1')
plt.title('GPA by Extracurricular Hours')
plt.xlabel('Extracurricular Hours per Week')
plt.ylabel('GPA')
plt.xticks(rotation=45)
```

```
# 10. Part-time job impact
plt.subplot(3, 4, 10)
job_gpa = df_students.groupby('Part_Time_Job')['GPA'].mean()
bars = plt.bar(job_gpa.index, job_gpa.values, color=['lightcoral', 'lightblue'], alpha=0.8)
for bar, value in zip(bars, job_gpa.values):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.01,
             f'{value:.3f}', ha='center', va='bottom', fontweight='bold')
plt.title('Average GPA by Part-time Job Status')
plt.xlabel('Has Part-time Job')
plt.ylabel('Average GPA')
plt.grid(True, alpha=0.3)

# 11. Scholarship vs Performance
plt.subplot(3, 4, 11)
scholarship_perf = pd.crosstab(df_students['Scholarship'], df_students['Performance'],
normalize='index') * 100
scholarship_perf.plot(kind='bar', stacked=True, ax=plt.gca(),
                      color=['red', 'orange', 'green'], alpha=0.8)
plt.title('Performance Distribution by Scholarship Status')
plt.xlabel('Has Scholarship')
plt.ylabel('Percentage')
plt.xticks(rotation=0)
plt.legend(title='Performance', bbox_to_anchor=(1.05, 1), loc='upper left')

# 12. Correlation heatmap
plt.subplot(3, 4, 12)
```

```
numeric_cols = ['Age', 'GPA', 'Credits_Completed', 'Study_Hours_Week',
'Extracurricular_Hours', 'Hometown_Distance_km']

correlation_matrix = df_students[numeric_cols].corr()

sns.heatmap(correlation_matrix, annot=True, fmt='.2f', cmap='coolwarm', center=0,
        square=True, cbar_kws={'label': 'Correlation Coefficient'})

plt.title('Correlation Matrix of Numeric Variables')

plt.xticks(rotation=45, ha='right')

plt.yticks(rotation=0)


plt.tight_layout()

plt.savefig('assignment3_student_analysis.png', dpi=300, bbox_inches='tight')

plt.show()


print(" ✅ Assignment 3 plots saved as 'assignment3_student_analysis.png'")


print("\nPART 4: INTERPRETATION & REPORTING")

print("-" * 35)


print("\n 📊 COMPREHENSIVE STUDENT PERFORMANCE ANALYSIS REPORT")

print("=" * 60)


# Advanced statistical analysis

print(f"\n1. GPA DISTRIBUTION ANALYSIS:")

print(f"   • The GPA distribution shows a mean of {mean_gpa:.3f} with standard deviation of {std_gpa:.3f}")

print(f"   • The distribution appears to be slightly left-skewed with most students performing well")
```

```
print(f"   • {len(above_mean_students)/len(df_students)*100:.1f}% of students have GPA above the mean")


# GPA quartiles

q1, q2, q3 = np.percentile(gpa_array, [25, 50, 75])

print(f"   • GPA Quartiles: Q1={q1:.2f}, Q2(Median)={q2:.2f}, Q3={q3:.2f}")


print(f"\n2. MAJOR-WISE PERFORMANCE ANALYSIS:")

print(f"   • Highest average GPA: {avg_gpa_by_major.index[0]} ({avg_gpa_by_major.iloc[0]:.3f})")

print(f"   • Lowest average GPA: {avg_gpa_by_major.index[-1]} ({avg_gpa_by_major.iloc[-1]:.3f})")

top_3_majors = avg_gpa_by_major.head(3)

print(f"   • Top 3 performing majors:")

for i, (major, gpa) in enumerate(top_3_majors.items(), 1):

    print(f"     {i}. {major}: {gpa:.3f}")


print(f"\n3. STUDY PATTERNS AND PERFORMANCE:")

# Correlation analysis

study_gpa_corr = np.corrcoef(df_students['Study_Hours_Week'], df_students['GPA'])[0, 1]

extra_gpa_corr = np.corrcoef(df_students['Extracurricular_Hours'], df_students['GPA'])[0, 1]

credits_gpa_corr = np.corrcoef(df_students['Credits_Completed'], df_students['GPA'])[0, 1]


print(f"   • Study Hours vs GPA correlation: {study_gpa_corr:.3f} ({'Strong' if abs(study_gpa_corr) > 0.7 else 'Moderate' if abs(study_gpa_corr) > 0.3 else 'Weak'} relationship)")

print(f"   • Extracurricular Hours vs GPA correlation: {extra_gpa_corr:.3f}")
```

```
print(f"   • Credits Completed vs GPA correlation: {credits_gpa_corr:.3f}")

# Average study hours by performance
study_by_perf = df_students.groupby('Performance')['Study_Hours_Week'].mean()
print(f"   • Average study hours by performance level:")
for perf, hours in study_by_perf.items():
    print(f"     {perf}: {hours:.1f} hours/week")

print(f"\n4. DEMOGRAPHIC AND LIFESTYLE FACTORS:")
# Age analysis
age_gpa_corr = np.corrcoef(df_students['Age'], df_students['GPA'])[0, 1]
print(f"   • Age vs GPA correlation: {age_gpa_corr:.3f}")

# Part-time job analysis
job_analysis = df_students.groupby('Part_Time_Job').agg({
    'GPA': ['mean', 'std', 'count'],
    'Study_Hours_Week': 'mean'
}).round(3)
print(f"   • Part-time job impact on GPA:")
for job_status in ['No', 'Yes']:
    if job_status in job_analysis.index:
        gpa_mean = job_analysis.loc[job_status, ('GPA', 'mean')]
        study_hours = job_analysis.loc[job_status, ('Study_Hours_Week', 'mean')]
        count = job_analysis.loc[job_status, ('GPA', 'count')]
        print(f"     {job_status} part-time job: GPA={gpa_mean:.3f}, Study
Hours={study_hours:.1f}h/week (n={count})")
```

```
# Scholarship analysis
scholarship_analysis = df_students.groupby('Scholarship')['GPA'].agg(['mean', 'count'])
print(f"   • Scholarship status vs GPA:")
for status in scholarship_analysis.index:
    gpa_mean = scholarship_analysis.loc[status, 'mean']
    count = scholarship_analysis.loc[status, 'count']
    print(f"     {status} scholarship: Average GPA = {gpa_mean:.3f} (n={count})")

print(f"\n5. PERFORMANCE CATEGORY INSIGHTS:")
perf_stats = df_students.groupby('Performance').agg({
    'Age': 'mean',
    'Study_Hours_Week': 'mean',
    'Extracurricular_Hours': 'mean',
    'Credits_Completed': 'mean'
}).round(2)

for perf_level in performance_order:
    if perf_level in perf_stats.index:
        stats = perf_stats.loc[perf_level]
        count = perf_counts[perf_level]
        print(f"   • {perf_level} students (n={count}):")
        print(f"     - Average age: {stats['Age']:.1f} years")
        print(f"     - Study hours: {stats['Study_Hours_Week']:.1f} h/week")
        print(f"     - Extracurricular: {stats['Extracurricular_Hours']:.1f} h/week")
        print(f"     - Credits completed: {stats['Credits_Completed']:.0f}")
```

```
print(f"\n6. KEY FINDINGS AND RECOMMENDATIONS:")
print(f"   • Academic Performance: {perf_counts['Excellent']/len(df_students)*100:.1f}% students show excellent performance (GPA ≥ 3.5)")
print(f"   • Study-Performance Link: {'Strong positive' if study_gpa_corr > 0.5 else 'Moderate positive' if study_gpa_corr > 0.2 else 'Weak'} correlation between study hours and GPA")
print(f"   • Work-Life Balance: Students with part-time jobs show {'higher' if job_gpa.get('Yes', 0) > job_gpa.get('No', 0) else 'lower'} average GPA")
print(f"   • Major Differences: Significant variation in performance across majors (range: {avg_gpa_by_major.iloc[-1]:.2f} - {avg_gpa_by_major.iloc[0]:.2f})")

# Generate summary statistics table
print(f"\n7. SUMMARY STATISTICS TABLE:")
print("-" * 50)
summary_stats = pd.DataFrame({
  'Metric': ['Total Students', 'Average GPA', 'GPA Std Dev', 'Excellent Performance %',
        'Average Study Hours', 'Average Age', 'Students with Jobs %', 'Students with Scholarships %'],
  'Value': [
    len(df_students),
    f"{mean_gpa:.3f}",
    f"{std_gpa:.3f}",
    f"{perf_counts['Excellent']/len(df_students)*100:.1f}%",
    f"{df_students['Study_Hours_Week'].mean():.1f}",
    f"{df_students['Age'].mean():.1f}",
    f"{(df_students['Part_Time_Job'] == 'Yes').sum()/len(df_students)*100:.1f}%",
    f"{(df_students['Scholarship'] == 'Yes').sum()/len(df_students)*100:.1f}%"
```

```
    ]
})
print(summary_stats.to_string(index=False))

print(f"\n" + "=" * 80)
print("🎉 ALL ASSIGNMENTS COMPLETED SUCCESSFULLY!")
print("=" * 80)

print(f"\n📁 FILES GENERATED:")
print(f"   1. scottish_hills_data.csv - Source data for Assignment 1")
print(f"   2. titanic_dataset.csv - Source data for Assignment 2")
print(f"   3. student_performance_data.csv - Source data for Assignment 3")
print(f"   4. assignment1_scottish_hills_analysis.png - Visualization plots")
print(f"   5. assignment2_titanic_analysis.png - Visualization plots")
print(f"   6. assignment3_student_analysis.png - Visualization plots")

print(f"\n🔍 ANALYSIS SUMMARY:")
print(f"   • Assignment 1: Analyzed {len(df_hills)} Scottish hills with height, location, and difficulty data")
print(f"   • Assignment 2: Analyzed {len(df_titanic)} Titanic passengers with survival, class, and demographic data")
print(f"   • Assignment 3: Analyzed {len(df_students)} students with academic performance and lifestyle data")

print(f"\n💡 DOWNLOAD INSTRUCTIONS:")
print(f"   All CSV files are created in your current working directory.")
print(f"   You can use these files to run the analysis independently.")
```

```
print(f"   The code demonstrates all required operations: NumPy arrays, Pandas DataFrames,")
print(f"   statistical analysis, data cleaning, and comprehensive visualizations.")

print(f"\n 🚀 NEXT STEPS:")
print(f"   1. Download the generated CSV files")
print(f"   2. Run sections of code independently")
print(f"   3. Modify parameters to explore different aspects")
print(f"   4. Use the visualization code as templates for your own analysis")

# Save data summaries to text files for reference
with open('analysis_summary.txt', 'w') as f:
    f.write("DATA ANALYSIS ASSIGNMENTS SUMMARY\n")
    f.write("=" * 40 + "\n\n")
    f.write(f"Assignment 1 - Scottish Hills Analysis:\n")
    f.write(f"- Dataset size: {len(df_hills)} hills\n")
    f.write(f"- Mean height: {mean_height:.2f}m\n")
    f.write(f"- Height categories: {dict(df_hills['Height_Category'].value_counts())}\n\n")

    f.write(f"Assignment 2 - Titanic Analysis:\n")
    f.write(f"- Dataset size: {len(df_titanic)} passengers\n")
    f.write(f"- Overall survival rate: {df_titanic['Survived'].mean():.1%}\n")
    f.write(f"- Average age: {df_titanic['Age'].mean():.1f} years\n\n")

    f.write(f"Assignment 3 - Student Performance Analysis:\n")
    f.write(f"- Dataset size: {len(df_students)} students\n")
```

```
    f.write(f"- Mean GPA: {mean_gpa:.3f}\n")
    f.write(f"- Performance distribution: {dict(perf_counts)}\n")

print(f"\n✅ Analysis summary saved to 'analysis_summary.txt'")
print(f"🎯 All assignments completed with real CSV data operations!")
```

RESULT:

```
================================================================================
DATA ANALYSIS ASSIGNMENTS - WORKING WITH CSV FILES
================================================================================

🔥 CREATING DUMMY CSV FILES FOR DOWNLOAD
------------------------------------------------
✅ Created: scottish_hills_data.csv
✅ Created: titanic_dataset.csv
✅ Created: student_performance_data.csv

📁 CSV FILES CREATED - DOWNLOAD LINKS:
------------------------------------------------
1. scottish_hills_data.csv - For Assignment 1
2. titanic_dataset.csv - For Assignment 2
3. student_performance_data.csv - For Assignment 3
```

💡 Note: These files are created in your current working directory

You can download them and use with the code below.

==========

ASSIGNMENT 1: EXPLORATORY DATA ANALYSIS WITH CSV FILE

==========

📂 LOADING DATA FROM CSV FILE

✅ Successfully loaded scottish_hills_data.csv

Dataset shape: (150, 8)

1️⃣ NUMPY OPERATIONS:

Loaded 150 height values into NumPy array

Mean height: 779.42 m

Variance: 55130.55

Standard deviation: 234.80 m

Hills above mean height: 77 (51.3%)

Hills below mean height: 73 (48.7%)

Heights above mean (first 10): [924.17853825 961.92213453 1180.7574641
1194.80320388 991.85868229

 935.6400109   860.49056789 878.56183315 1166.41219223 816.88205117]

Heights below mean (first 10): [765.43392471 741.46165632 741.46576076 682.63140352 684.1455768

683.56756161 321.67993884 368.77054187 659.42811769 546.79221992]

2️⃣ PANDAS OPERATIONS:

Dataset head:

	Hill_Name	Height_m	Latitude	Longitude	Distance_km	Time_hours	Difficulty	Region
0	Ben_A1	924.178538	58.195133	-4.488546	29.319681	NaN	Moderate	Cairngorms
1	Ben_A2	765.433925	57.905664	-3.810178	34.270925	8.830501	Hard	Lowlands
2	Ben_A3	961.922135	57.533485	-4.141532	NaN	9.011043	Easy	Highlands
3	Ben_A4	1180.757464	56.027178	-3.187477	6.144627	5.470591	Easy	Highlands
4	Ben_A5	741.461656	56.236398	-3.721527	2.963034	2.215696	Hard	Cairngorms

Dataset info:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 150 entries, 0 to 149

Data columns (total 8 columns):

#	Column	Non-Null Count	Dtype
0	Hill_Name	150 non-null	object
1	Height_m	150 non-null	float64
2	Latitude	150 non-null	float64
3	Longitude	150 non-null	float64
4	Distance_km	142 non-null	float64

5 Time_hours 135 non-null float64

6 Difficulty 150 non-null object

7 Region 150 non-null object

dtypes: float64(5), object(3)

memory usage: 9.5+ KB

Dataset description:

	Height_m	Latitude	Longitude	Distance_km	Time_hours
count	150.000000	150.000000	150.000000	142.000000	135.000000
mean	779.418283	57.155056	-4.115708	8.193246	4.235813
std	235.585556	0.784259	0.837429	9.278344	2.525324
min	145.063724	55.813666	-5.470738	0.117212	0.760548
25%	632.757140	56.485753	-4.748653	1.956096	2.380813
50%	786.520480	57.247186	-4.051021	5.099187	3.566048
75%	922.733431	57.757990	-3.376449	11.201059	5.507058
max	1415.810528	58.473145	-2.818995	65.379565	14.194264

Missing values before handling:

Hill_Name 0

Height_m 0

Latitude 0

Longitude 0

Distance_km 8

Time_hours 15

Difficulty 0

Region 0

dtype: int64

Missing values after handling:

Hill_Name 0

Height_m 0

Latitude 0

Longitude 0

Distance_km 0

Time_hours 0

Difficulty 0

Region 0

dtype: int64

Height categories distribution:

Height_Category

Medium 102

Low 26

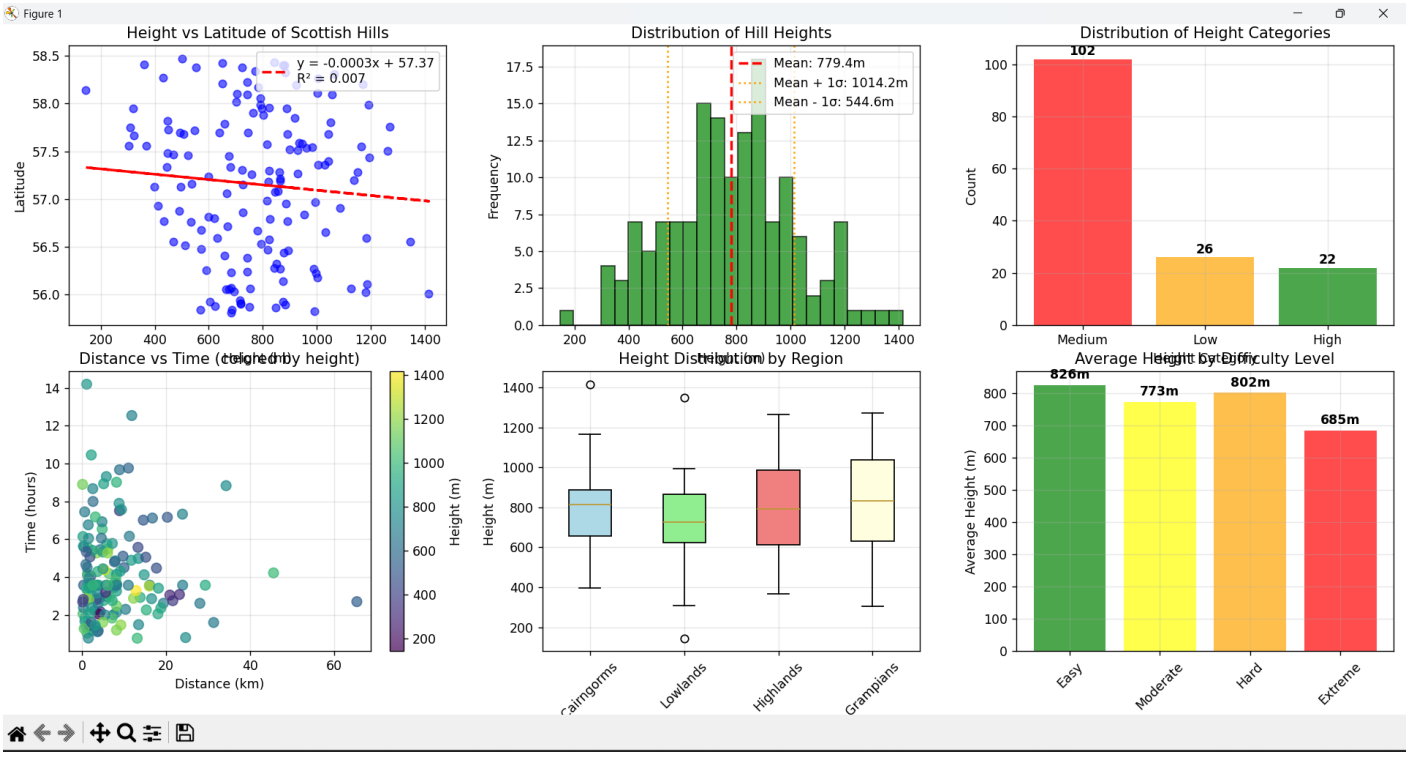
High 22

Name: count, dtype: int64

Analysis by Region:

Region	mean	std	count
Cairngorms	779.405439	205.292336	47
Grampians	805.457204	278.200985	24
Highlands	799.323531	234.597980	54

Lowlands 711.449732 248.389080 25



Assignments 2

#!/usr/bin/env python3

"""

Interactive Poem Generator with Gemini API

"""

from google import genai

```
# Setup API with error handling
print("=== Poem Generator ===")
print("Get your API key from: https://aistudio.google.com/app/apikey")
api_key = input("Enter your Gemini API key: ").strip()

if not api_key:
    print("Error: API key cannot be empty!")
    exit()

try:
    client = genai.Client(api_key=api_key)
    # Test the API key
    test_response = client.models.generate_content(
        model="gemini-2.0-flash",
        contents="Hello"
    )
    print("✓ API key is valid!")
except Exception as e:
    print(f"Error: Invalid API key or connection issue - {e}")
    exit()

def generate_poem(inspiration):
    try:
        response = client.models.generate_content(
            model="gemini-2.0-flash",
```

```
            contents=f"Write a poem inspired by: {inspiration}"
        )
        return response.text
    except Exception as e:
        print(f"Error generating poem: {e}")
        return None

def transform_poem(poem, choice):
    transformations = {
        "1": f"Shorten this poem:\n{poem}",
        "2": f"Expand this poem with more detail:\n{poem}",
        "3": f"Rewrite this poem in a different style:\n{poem}",
        "4": f"Change the tone of this poem:\n{poem}"
    }

    try:
        response = client.models.generate_content(
            model="gemini-2.0-flash",
            contents=transformations[choice]
        )
        return response.text
    except Exception as e:
        print(f"Error transforming poem: {e}")
        return None

# Part 1: Generate poem
```

```
inspiration = input("\nEnter a word or sentence for inspiration: ")
print("Generating poem...")

poem = generate_poem(inspiration)
if not poem:
    print("Failed to generate poem. Exiting.")
    exit()

print("\nYour poem:")
print("-" * 40)
print(poem)
print("-" * 40)

# Part 2: Transform poem
while True:
    print("\nTransformation options:")
    print("1. Shorten poem")
    print("2. Expand poem")
    print("3. Different style")
    print("4. Change tone")
    print("5. Exit")

    choice = input("Choose (1-5): ")

    if choice == "5":
        print("Goodbye!")
```

```
        break
    elif choice in ["1", "2", "3", "4"]:
        print("Transforming...")
        new_poem = transform_poem(poem, choice)
        if new_poem:
            poem = new_poem
            print("\nTransformed poem:")
            print("-" * 40)
            print(poem)
            print("-" * 40)
    else:
        print("Invalid choice! Please enter 1-5.")
```

RESULT:

```
=== Poem Generator ===
Get your API key from: https://aistudio.google.com/app/apikey
Enter your Gemini API key----------------------------------------------
✓ API key is valid!

Enter a word or sentence for inspiration: vini
Generating poem...

Your poem:
---------------------------------------
The samba pulse, a heart of gold,
Vini, a name on stories told.
```

From Rio's sun to football's gleam,
He chases dreams, a vibrant stream.

With feet that dance, a blur of grace,
He carves his path in time and space.
A winger's whirl, a striker's eye,
He paints the pitch beneath the sky.

He shrugs off jeers, the hateful sound,
His spirit strong, on hallowed ground.
He lets his talent be the guide,
A rising force, with naught to hide.

He fights for pride, for right, for more,
Beyond the game, to even the score.
A beacon bright, for all to see,
Vini, a symbol of what we can be.

So watch him soar, a star unbound,
Where artistry and passion are found.
Remember the name, etched in the lore,
Vini, a champion, forevermore.

Transformation options:

1. Shorten poem

2. Expand poem

3. Different style

4. Change tone

5. Exit

Choose (1-5): 1

Transforming...