HW1

Problem1

a

```python
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt

df = pd.read_csv("CAPOP.csv")

# Convert the date column to extract the year
df["Year"] = pd.to_datetime(df["observation_date"]).dt.year

# Drop the 2024 data point
df = df[df["Year"] < 2024]

# Define the independent (t) and dependent (y) variables
X = df["Year"]
y = df["CAPOP"]

# Add a constant for the intercept
X = sm.add_constant(X)

# Fit the linear regression model
model = sm.OLS(y, X).fit()

# Get the parameter estimates and confidence intervals
beta0, beta1 = model.params
conf_int = model.conf_int()
```

```python
# Display results
print(f"Estimated beta0 (Intercept): {beta0:.2f}")
print(f"Estimated beta1 (Slope): {beta1:.2f}")
print(f"95% Confidence Interval for beta0: ({conf_int.iloc[0, 0]:.2f},\
         {conf_int.iloc[0, 1]:.2f})")
print(f"95% Confidence Interval for beta1: ({conf_int.iloc[1, 0]:.2f},\
         {conf_int.iloc[1, 1]:.2f})")
```

```
Estimated beta0 (Intercept): -686337.86
Estimated beta1 (Slope): 359.14
95% Confidence Interval for beta0: (-707361.56,         -665314.16)
95% Confidence Interval for beta1: (348.43,         369.86)
```

Interpretation of Regression Estimates

In the fitted linear model:

[$y_t = \beta_0 + \beta_1 t + \epsilon_t$]

where:

- (y_t) represents the **annual resident population of California** (in thousands),
- (t) represents the **year**,
- (β_0) is the **intercept**, which theoretically represents the population at ($t = 0$) (Year 0 AD),
- (β_1) is the **slope**, indicating the **average annual population growth rate**.

Interpretation:

1. Intercept (β_0)

The estimated intercept is: [$\hat{\beta}_0 = -686300$] This implies that, if extrapolated to **Year 0**, the estimated population would be **-686.3 million**, which is clearly **nonsensical**. This happens because:

- The model is fitted using data from **1900 to 2023**, and extrapolating too far beyond the observed range is unreliable.
- The intercept is mathematically necessary for the regression equation but does **not** have a meaningful real-world interpretation in this context.

2. Slope (β_1)

The estimated slope is: [$\hat{\beta}_1 = 359.14$]

This means that **California's population increases by an average of 359,140 people per year**. This estimate is reasonable because:

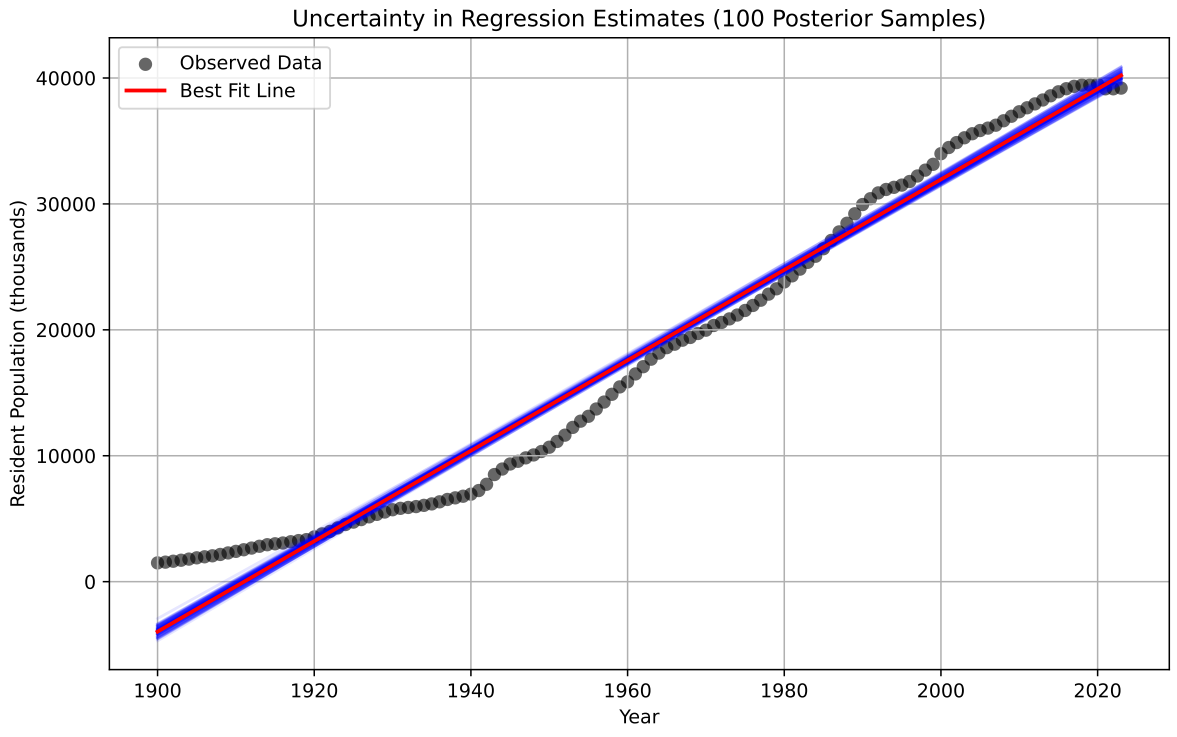
- Historical data shows **consistent population growth** from 1900 2023.
- The high **R² value (0.973)** suggests that the model captures a strong linear trend in population growth.

Conclusion:

- (β_1) **provides a meaningful estimate of average annual growth.**
- (β_0) **has no real-world interpretation but is mathematically required.**
- **The model is useful for understanding historical trends but may need refinement for future projections.**

b

```
# Extract estimated parameters
beta0_hat, beta1_hat = model.params
cov_matrix = model.cov_params()

# Sample 100 (beta0, beta1) pairs from posterior
np.random.seed(42)
posterior_samples = np.random.multivariate_normal(mean=model.params,\
                 cov=cov_matrix, size=100)

# Plot observed data
plt.figure(figsize=(10, 6))
plt.scatter(df["Year"], df["CAPOP"], color="black", \
                label="Observed Data", alpha=0.6)

# Generate lines for posterior samples
years = np.linspace(df["Year"].min(), df["Year"].max(), 100)
for beta0, beta1 in posterior_samples:
    plt.plot(years, beta0 + beta1 * years, color="blue", alpha=0.1)

# Plot best-fit regression line
plt.plot(years, beta0_hat + beta1_hat * years, color="red",\
             label="Best Fit Line", linewidth=2)

# Labels and legend
plt.xlabel("Year")
plt.ylabel("Resident Population (thousands)")
plt.title("Uncertainty in Regression Estimates (100 Posterior Samples)")
plt.legend()
plt.grid(True)
plt.show()
```



The blue lines are closely clustered around the red best-fit line, it suggests that our model is highly confident in its parameter estimates.

c

```
# Predict for 2024
X_2024 = pd.DataFrame({"Year": [2024]})
X_2024 = sm.add_constant(X_2024, has_constant='add')
pred_2024 = model.get_prediction(X_2024)
```

```python
# Extract mean prediction and confidence intervals
mean_prediction = pred_2024.predicted_mean[0]
conf_int = pred_2024.conf_int(alpha=0.05)  # 95% confidence interval
lower_bound, upper_bound = conf_int[0]

# Display results
print(f"Predicted 2024 Population: {mean_prediction:.2f} (in thousands)")
print(f"95% Confidence Interval: ({lower_bound:.2f}, {upper_bound:.2f})")
```

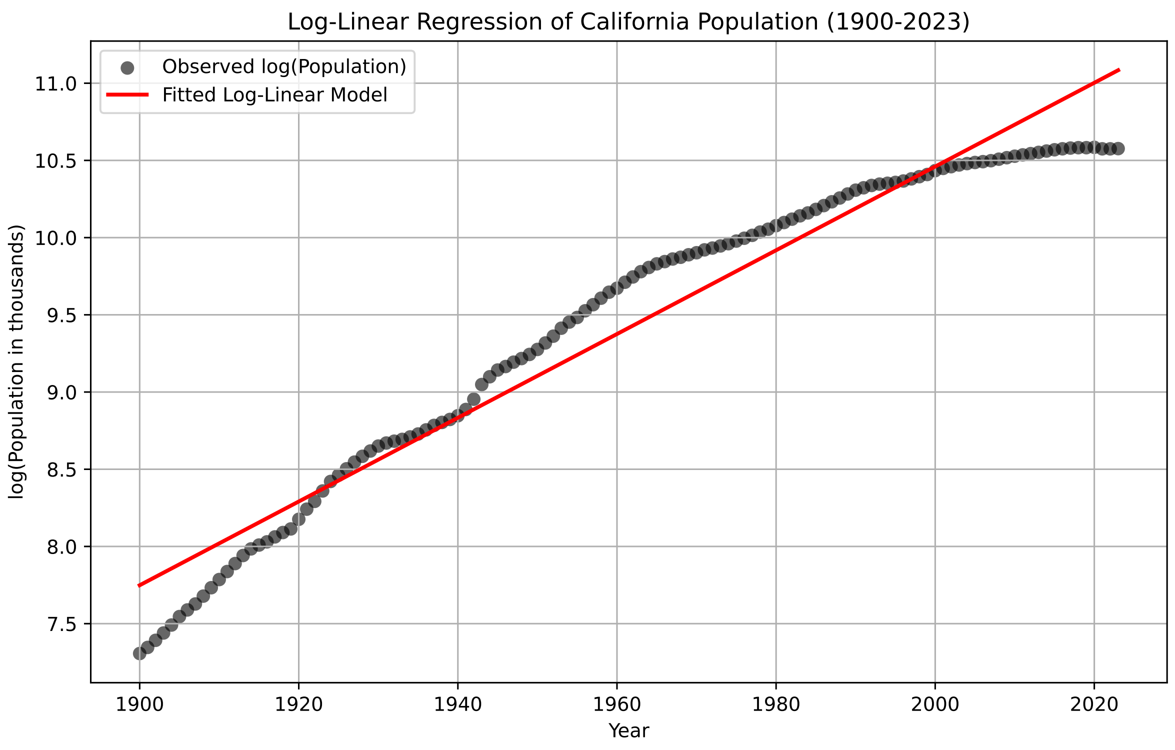
```
Predicted 2024 Population: 40565.19 (in thousands)
95% Confidence Interval: (39793.35, 41337.03)
```

d

```python
# Load dataset
df = pd.read_csv("CAPOP.csv")

# Convert the date column to extract the year
df["Year"] = pd.to_datetime(df["observation_date"]).dt.year

# Drop the 2024 data point if present
df = df[df["Year"] < 2024]

# Define the independent (t) and dependent (log(y)) variables
X = df["Year"]
y = np.log(df["CAPOP"])

X = sm.add_constant(X)
model_log = sm.OLS(y, X).fit()

# Get the parameter estimates and confidence intervals
beta0_log, beta1_log = model_log.params
conf_int_log = model_log.conf_int()

# Display results
print(f"Estimated Beta0 (Intercept): {beta0_log:.4f}")
print(f"Estimated Beta1 (Slope): {beta1_log:.4f}")
print(f"95% Confidence Interval for Beta0: ({conf_int_log.iloc[0, 0]:.4f},\
         {conf_int_log.iloc[0, 1]:.4f})")
```

```python
print(f"95% Confidence Interval for Beta1: ({conf_int_log.iloc[1, 0]:.4f},\
         {conf_int_log.iloc[1, 1]:.4f})")

# Plot observed vs fitted values
plt.figure(figsize=(10, 6))
plt.scatter(df["Year"], np.log(df["CAPOP"]), color="black",\
         label="Observed log(Population)", alpha=0.6)
plt.plot(df["Year"], model_log.fittedvalues, color="red",\
         label="Fitted Log-Linear Model", linewidth=2)

# Labels and legend
plt.xlabel("Year")
plt.ylabel("log(Population in thousands)")
plt.title("Log-Linear Regression of California Population (1900-2023)")
plt.legend()
plt.grid(True)
plt.show()
```

```
Estimated Beta0 (Intercept): -43.7710
Estimated Beta1 (Slope): 0.0271
95% Confidence Interval for Beta0: (-45.9169,         -41.6250)
95% Confidence Interval for Beta1: (0.0260,         0.0282)
```



Interpretation of Regression Estimates

Log-Linear Regression Interpretation:

- (β_1) **(slope)** represents the **logarithmic change in population growth rate.**
- (β_0) **(intercept)** theoretically reprzesents the **log of the initial population at** $t = 0$, but usually has no practical significance.
- A **high** R^2 **value** indicates that the model fits the data well.

e

```
from scipy.stats import norm

# Extract posterior mean and variance of beta1
beta1_hat = model_log.params["Year"]
beta1_std = model_log.bse["Year"]

# Compute posterior probability P(beta1 > 0.03)
```

```
p_beta1_greater_003 = 1 - norm.cdf(0.03, loc=beta1_hat, scale=beta1_std)

# Display result
print(f"Posterior Probability P(beta1 > 0.03): {p_beta1_greater_003:.4f}")
```

```
Posterior Probability P(beta1 > 0.03): 0.0000
```

f

```
# Predict for 2024 (log scale)
X_2024 = pd.DataFrame({"Year": [2024]})
X_2024 = sm.add_constant(X_2024, has_constant="add")
log_pred_2024 = model_log.get_prediction(X_2024)

# Extract mean prediction and confidence intervals (log scale)
mean_log_pred = log_pred_2024.predicted_mean[0]
conf_int_log = log_pred_2024.conf_int(alpha=0.05)
lower_log, upper_log = conf_int_log[0]

# Convert back to population scale (exponential transformation)
mean_pred_2024 = np.exp(mean_log_pred)
lower_pred_2024 = np.exp(lower_log)
upper_pred_2024 = np.exp(upper_log)

# Display results
print(f"Predicted 2024 Population: {mean_pred_2024:.2f} (in thousands)")
print(f"95% Confidence Interval: ({lower_pred_2024:.2f}, \
                                    {upper_pred_2024:.2f})")
```

```
Predicted 2024 Population: 66909.89 (in thousands)
95% Confidence Interval: (61840.83,                                    72394.45)
```

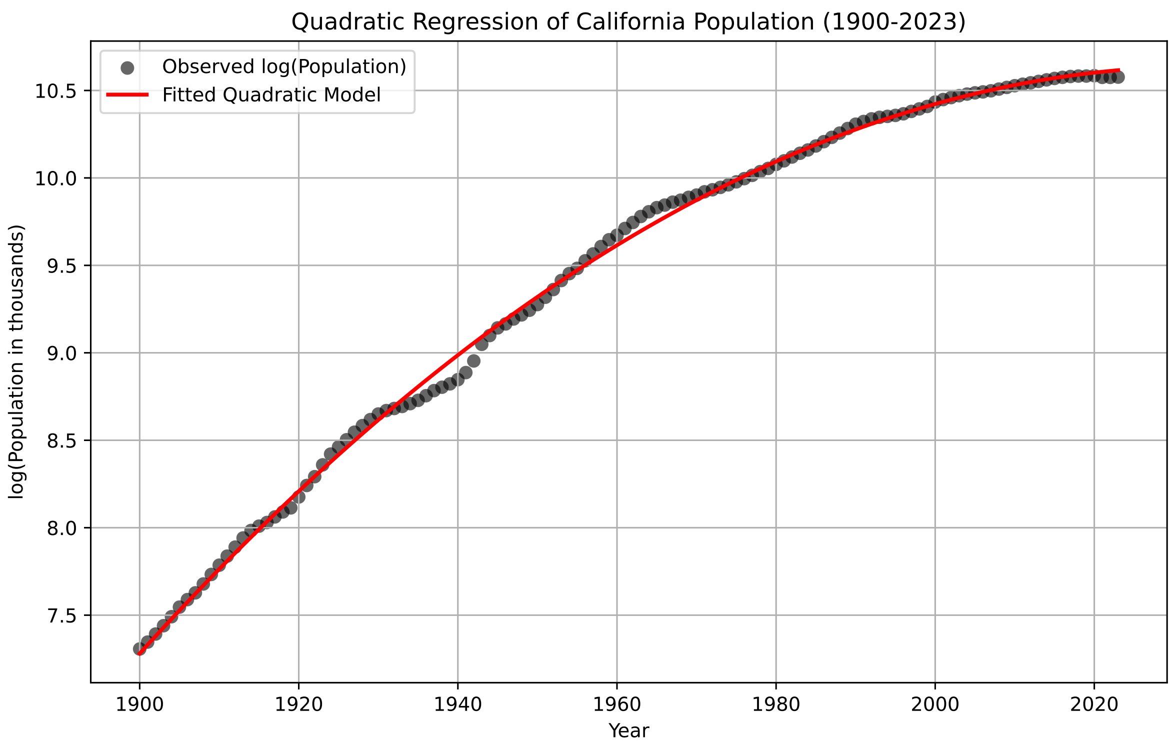
g

```
# Load dataset
df = pd.read_csv("CAPOP.csv")
```

```
# Convert the date column to extract the year
df["Year"] = pd.to_datetime(df["observation_date"]).dt.year

# Drop the 2024 data point if present
df = df[df["Year"] < 2024]

# Define the independent (t, t²) and dependent (log(y)) variables
X = df[["Year"]].copy()
X["Year^2"] = X["Year"] ** 2
y = np.log(df["CAPOP"])

X = sm.add_constant(X)
model_quad = sm.OLS(y, X).fit()

# Extract estimated parameters (beta0, beta1, beta2)
beta0_quad, beta1_quad, beta2_quad = model_quad.params
conf_int_quad = model_quad.conf_int()

# Print estimated coefficients and confidence intervals
print(f"Estimated beta0 (Intercept): {beta0_quad:.4f}")
print(f"Estimated beta1 (Linear Term): {beta1_quad:.4f}")
print(f"Estimated beta2 (Quadratic Term): {beta2_quad:.4f}")

print(f"95% Confidence Interval for beta0: ({conf_int_quad.loc[\
    'const', 0]:.4f},{conf_int_quad.loc['const', 1]:.4f})")
print(f"95% Confidence Interval for beta1: ({conf_int_quad.loc[\
    'Year', 0]:.4f},{conf_int_quad.loc['Year', 1]:.4f})")
print(f"95% Confidence Interval for beta2: ({conf_int_quad.loc[\
    'Year^2', 0]:.6f},{conf_int_quad.loc['Year^2', 1]:.6f})")

# Predict for 2024 (log scale)
X_2024 = pd.DataFrame({"Year": [2024], "Year^2": [2024 ** 2]})
X_2024 = sm.add_constant(X_2024, has_constant="add")

log_pred_2024 = model_quad.get_prediction(X_2024)

# Extract mean prediction and confidence intervals (log scale)
mean_log_pred = log_pred_2024.predicted_mean[0]
conf_int_log = log_pred_2024.conf_int(alpha=0.05)
lower_log, upper_log = conf_int_log[0]

# Convert back to population scale (exponential transformation)
```

```python
mean_pred_2024 = np.exp(mean_log_pred)
lower_pred_2024 = np.exp(lower_log)
upper_pred_2024 = np.exp(upper_log)

# Display results
print(f"Predicted 2024 Population (Quadratic Model):\
                 {mean_pred_2024:.2f} (in thousands)")
print(f"95% Confidence Interval: ({lower_pred_2024:.2f},\
                 {upper_pred_2024:.2f})")

# Generate fitted values for the quadratic model
df["Fitted log(Population)"] = model_quad.fittedvalues

# Plot observed vs fitted values
plt.figure(figsize=(10, 6))
plt.scatter(df["Year"], np.log(df["CAPOP"]), color="black",\
             label="Observed log(Population)", alpha=0.6)
plt.plot(df["Year"], df["Fitted log(Population)"], color="red",\
             label="Fitted Quadratic Model", linewidth=2)

# Labels and legend
plt.xlabel("Year")
plt.ylabel("log(Population in thousands)")
plt.title("Quadratic Regression of California Population (1900-2023)")
plt.legend()
plt.grid(True)
plt.show()
```

```
Estimated beta0 (Intercept): -763.7776
Estimated beta1 (Linear Term): 0.7615
Estimated beta2 (Quadratic Term): -0.0002
95% Confidence Interval for beta0: (-788.7751,-738.7801)
95% Confidence Interval for beta1: (0.7360,0.7870)
95% Confidence Interval for beta2: (-0.000194,-0.000181)
Predicted 2024 Population (Quadratic Model):                 40933.48 (in thousands)
95% Confidence Interval: (40014.58,                 41873.47)
```



h

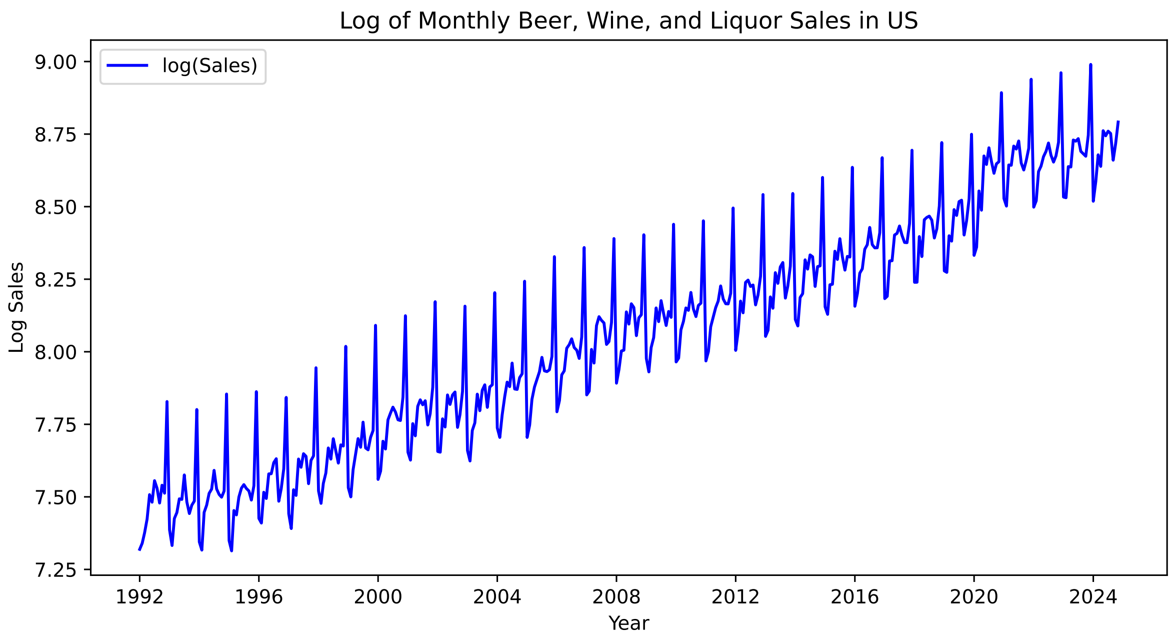
I would recommend model (3) for this dataset. Even though **model (1) has the smalled deviation**, **model (3) captures the overall population trends better.**

Problem 2

a

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

# load data
data = pd.read_csv("MRTSSM4453USN.csv")

data["observation_date"] = pd.to_datetime(data["observation_date"])
log_sales = np.log(data["MRTSSM4453USN"])
```

```
# plot
plt.figure(figsize=(10, 5))
plt.plot(data["observation_date"], log_sales,\
     label="log(Sales)", color="blue")
plt.xlabel("Year")
plt.ylabel("Log Sales")
plt.title("Log of Monthly Beer, Wine, and Liquor Sales in US")
plt.legend()
plt.show()
```

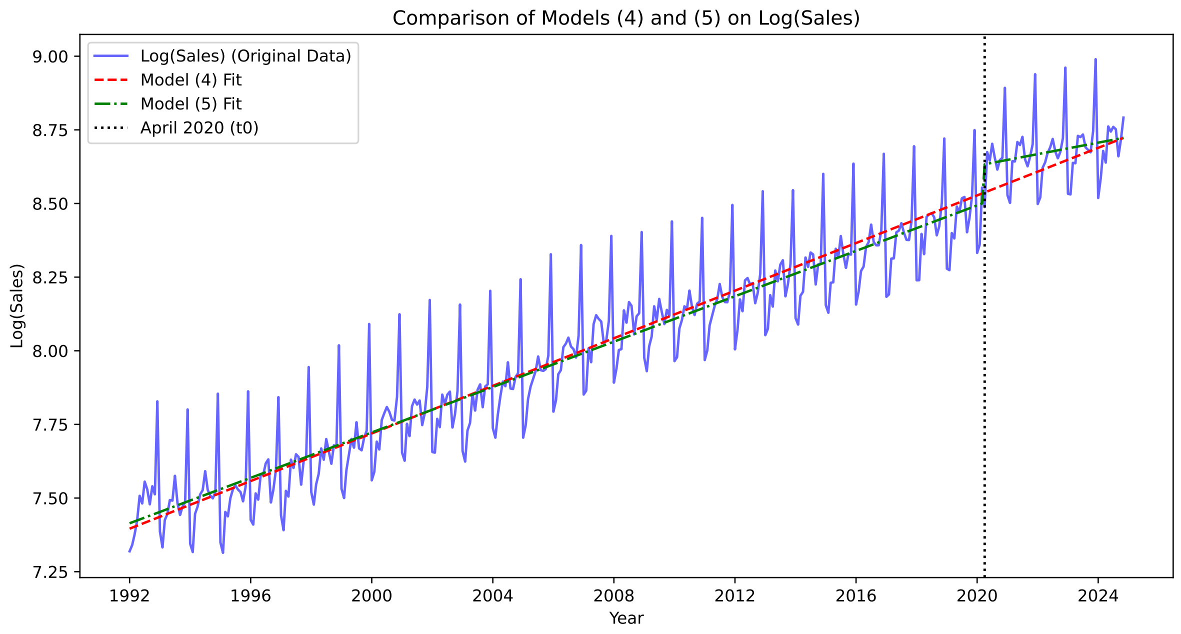


The sudden increase in the later part of the series suggests linear model may not be appropriate.

b

```
dates = data["observation_date"]
t = np.arange(len(dates))

# Find the index corresponding to April 2020
t0_index = np.where((dates.dt.year == 2020) & (dates.dt.month == 4))[0][0]
```

```
# Create indicator variables for structural change at t0
I_t0 = (t >= t0_index).astype(int) # t >= t0
t_I_t0 = t * I_t0  # t >= t0

# Fit model (4)
X4 = sm.add_constant(t)
model4 = sm.OLS(log_sales, X4).fit()

# Fit model (5)
X5 = sm.add_constant(np.column_stack([t, I_t0, t_I_t0]))
model5 = sm.OLS(log_sales, X5).fit()

# Generate predictions for both models
pred4 = model4.predict(X4)
pred5 = model5.predict(X5)

# Plot original data and fitted models
plt.figure(figsize=(12, 6))
plt.plot(dates, log_sales, label="Log(Sales) (Original Data)", color="blue",\
             alpha=0.6)
plt.plot(dates, pred4, label="Model (4) Fit", color="red", linestyle="--")
plt.plot(dates, pred5, label="Model (5) Fit", color="green", linestyle="-.")
plt.axvline(dates.iloc[t0_index], color="black", linestyle="dotted",\
             label="April 2020 (t0)")
plt.xlabel("Year")
plt.ylabel("Log(Sales)")
plt.title("Comparison of Models (4) and (5) on Log(Sales)")
plt.legend()
plt.show()
```



Model (5) better capture the sudden increase in rougly 2020, so model (5) is better.

c

```
import scipy.stats as stats
# Get parameter estimates and standard errors
beta2, beta3 = model5.params[2], model5.params[3]
se_beta3 = model5.bse[3]
se_beta2 = model5.bse[2]

# Compute P(beta3 < 0) using normal approximation
p_beta3_neg = stats.norm.cdf(0, loc=beta3, scale=se_beta3)

# Compute P(beta2 + beta3 * t0 > 0)
t0_value = t0_index  # Since t0 is in time index
mean_beta2_beta3 = beta2 + beta3 * t0_value
se_beta2_beta3 = np.sqrt(se_beta2**2 + (t0_value**2) * se_beta3**2)
p_beta2_beta3_pos = 1 - stats.norm.cdf(0, loc=mean_beta2_beta3,\
     scale=se_beta2_beta3)

# Print results
```

```
print(f"P(beta3 < 0) = {p_beta3_neg:.4f}")
print(f"P(beta2 + beta3 * t0 > 0) = {p_beta2_beta3_pos:.4f}")
```

```
P(beta3 < 0) = 0.9453
P(beta2 + beta3 * t0 > 0) = 0.6041
```

```
/tmp/ipykernel_584/3203333778.py:3: FutureWarning: Series.__getitem__ treating keys as posit
  beta2, beta3 = model5.params[2], model5.params[3]
/tmp/ipykernel_584/3203333778.py:4: FutureWarning: Series.__getitem__ treating keys as posit
  se_beta3 = model5.bse[3]
/tmp/ipykernel_584/3203333778.py:5: FutureWarning: Series.__getitem__ treating keys as posit
  se_beta2 = model5.bse[2]
```

d

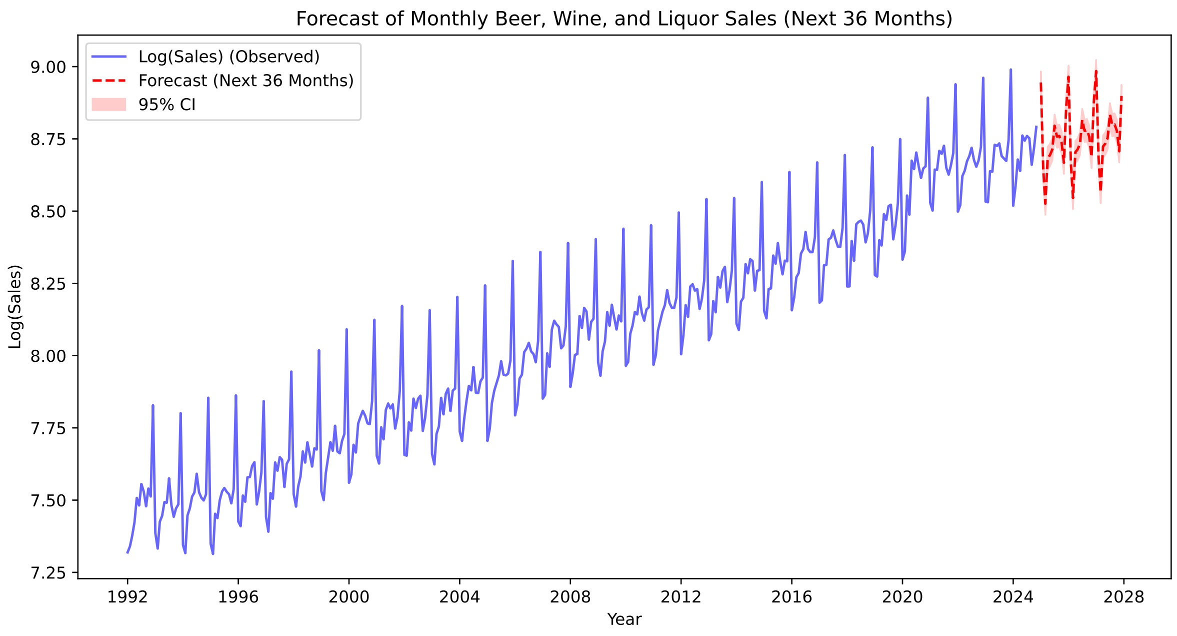
```
k = 4
# Generate Fourier terms
fourier_terms = []
for i in range(1, k + 1):
    fourier_terms.append(np.cos(2 * np.pi * i * t / 12))
    fourier_terms.append(np.sin(2 * np.pi * i * t / 12))

# Convert to NumPy array
fourier_terms = np.column_stack(fourier_terms)

# Construct the design matrix for Model (6)
X6 = sm.add_constant(np.column_stack([t, I_t0, t_I_t0, fourier_terms]))

# Fit the model
model6 = sm.OLS(log_sales, X6).fit()
```

I choose K=4, since there are 4 season in one year. Also, k=4 has the highest R^2 and lowest AIC

e

```python
# Extend time index for forecasting (36 future months)
t_future = np.arange(len(t), len(t) + 36)  # Future time indices
dates_future = pd.date_range(start=dates.iloc[-1], periods=37, freq='M')[1:]

# Generate Fourier terms for the future period
fourier_terms_future = []
for i in range(1, 5):  # k=4
    fourier_terms_future.append(np.cos(2 * np.pi * i * t_future / 12))
    fourier_terms_future.append(np.sin(2 * np.pi * i * t_future / 12))

fourier_terms_future = np.column_stack(fourier_terms_future)

# Generate future indicators for t0
I_t0_future = (t_future >= t0_index).astype(int)
t_I_t0_future = t_future * I_t0_future

# Construct future design matrix
X_future = sm.add_constant(np.column_stack([t_future, I_t0_future,\
     t_I_t0_future, fourier_terms_future]), has_constant='add')

# Predict future values using model (6)
pred_future = model6.predict(X_future)

# Calculate prediction confidence intervals (assuming normal errors)
se_pred = model6.bse.mean()  # Approximate standard error
ci_upper = pred_future + 1.96 * se_pred
ci_lower = pred_future - 1.96 * se_pred

# Plot observed data and forecast
plt.figure(figsize=(12, 6))
plt.plot(dates, log_sales, label="Log(Sales) (Observed)", color="blue",\
         alpha=0.6)
plt.plot(dates_future, pred_future, label="Forecast (Next 36 Months)",\
         color="red", linestyle="--")
plt.fill_between(dates_future, ci_lower, ci_upper, color="red", alpha=0.2,\
         label="95% CI")
plt.xlabel("Year")
plt.ylabel("Log(Sales)")
plt.title("Forecast of Monthly Beer, Wine, and Liquor Sales (Next 36 Months)")
plt.legend()
plt.show()
```

```
/tmp/ipykernel_584/2281166858.py:3: FutureWarning: 'M' is deprecated and will be removed in a
  dates_future = pd.date_range(start=dates.iloc[-1], periods=37, freq='M')[1:]
```



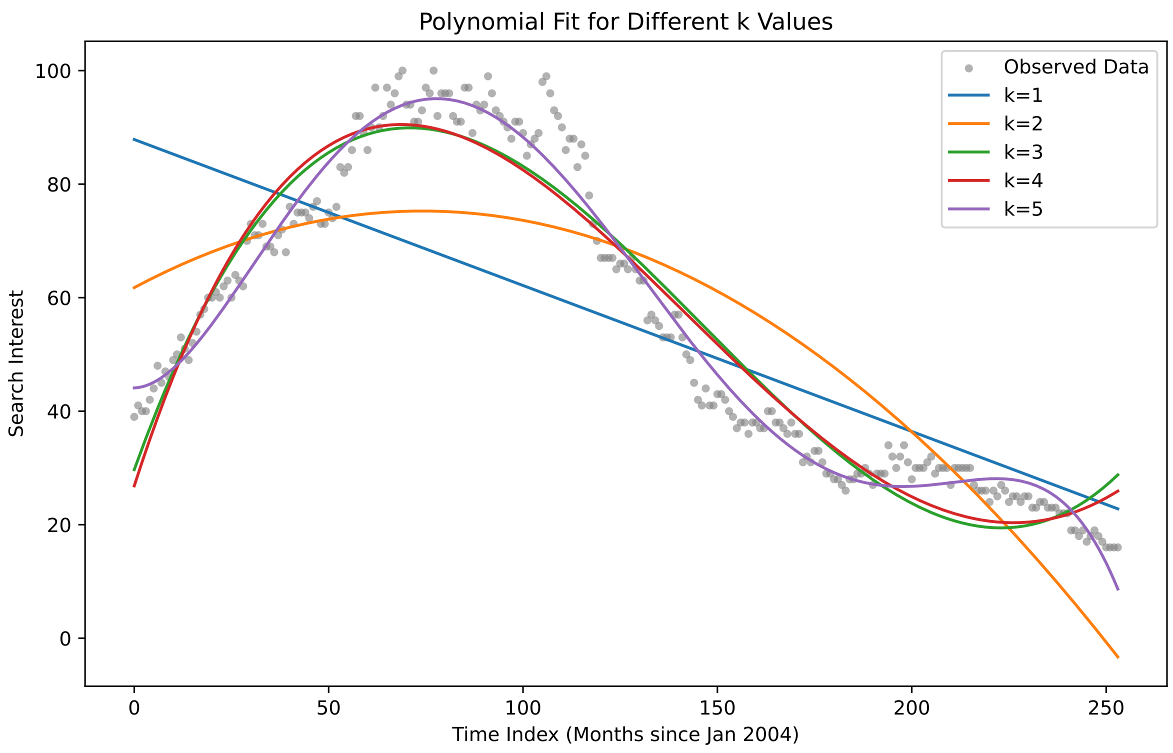
The model includes the piece-wise part after 2020, and chosed the best k to fit the model. I consider it a strong model. Thus, the prediction made by this model should be considered reasonable.

Problem 3

a

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Load the dataset
df = pd.read_csv("multiTimeline.csv", skiprows=1)
df.columns = ["Month", "Search Interest"]
```

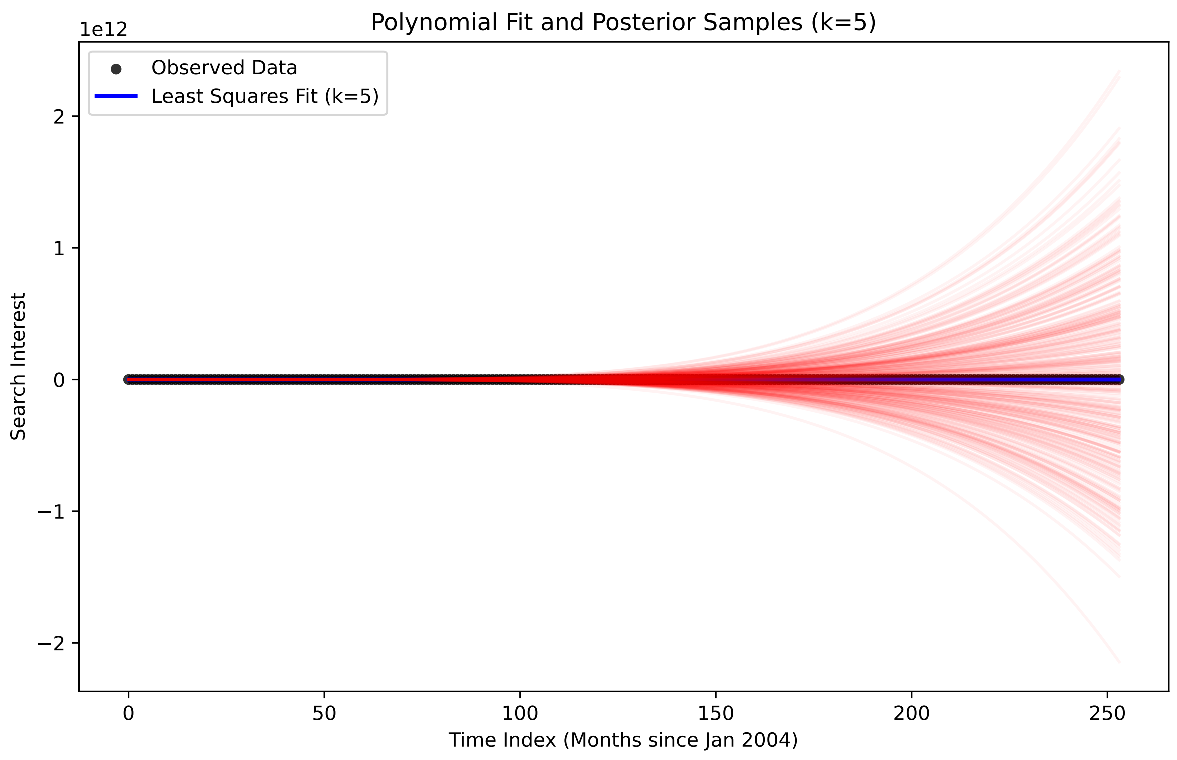
```python
# Convert "Month" column to datetime format and create a time index
df["Month"] = pd.to_datetime(df["Month"])
df["Time Index"] = np.arange(len(df))

# Extract feature (time index) and target variable (search interest)
X = df["Time Index"].values.reshape(-1, 1)
y = df["Search Interest"].values

# Define the maximum polynomial degree to evaluate
max_k = 5

# Initialize plot
plt.figure(figsize=(10, 6))
plt.scatter(df["Time Index"], y, label="Observed Data",\
     color="gray", s=10, alpha=0.6)

errors = []

# Fit polynomial regression models for k = 1 to max_k
for k in range(1, max_k + 1):
    poly = PolynomialFeatures(degree=k)
    X_poly = poly.fit_transform(X)

    # Fit least squares regression
    model = LinearRegression()
    model.fit(X_poly, y)
    y_pred = model.predict(X_poly)

    # Compute mean squared error
    mse = mean_squared_error(y, y_pred)
    errors.append(mse)

    # Plot the fitted polynomial curve
    plt.plot(df["Time Index"], y_pred, label=f"k={k}")

# Customize the plot
plt.xlabel("Time Index (Months since Jan 2004)")
plt.ylabel("Search Interest")
plt.title("Polynomial Fit for Different k Values")
plt.legend()
plt.show()
```



Amoung these 5 model, based on the graph I would say k=5 is the best fit. Since it has the samllest MSE and fit the trend of the data unlike k=3, 4.

b

```
# Fit the polynomial model with k=5
k_best = 5
poly = PolynomialFeatures(degree=k_best)
X_poly = poly.fit_transform(X)

# Fit least squares regression
model = LinearRegression()
model.fit(X_poly, y)
y_pred = model.predict(X_poly)

# Generate posterior samples
num_samples = 300
coef_samples = np.random.multivariate_normal(
    mean=model.coef_, cov=np.eye(len(model.coef_)) * 0.5, size=num_samples
)

# Generate polynomial curves from posterior samples
X_pred = np.linspace(X.min(), X.max(), 300).reshape(-1, 1)
X_pred_poly = poly.transform(X_pred)  # Transform new X values

posterior_curves = np.array([X_pred_poly @ coef for coef in coef_samples])

# Plot observed data
plt.figure(figsize=(10, 6))
plt.scatter(df["Time Index"], y, label="Observed Data",\
         color="black", s=20, alpha=0.8, marker="o")
plt.plot(df["Time Index"], y_pred, label=f"Least Squares Fit (k={k_best})",\
         color="blue", linewidth=2)
for i in range(num_samples):
    plt.plot(X_pred, posterior_curves[i], color="red", alpha=0.05)

plt.xlabel("Time Index (Months since Jan 2004)")
plt.ylabel("Search Interest")
plt.title(f"Polynomial Fit and Posterior Samples (k={k_best})")
plt.legend()
plt.show()
```



The red curves represent 300 samples from the posterior distribution of the polynomial coefficients. These curves show the range of possible trends given the uncertainty in the model parameters, revealing substantial uncertainty.