

## ✓ Interview Prep Simulation Scenarios 5

Your Name

## Getting Started

- Colab - get notebook from the gitmystuff DTSC5082 repository
- Save a Copy in Drive
- Rename the file using your name, just your name (Your Name.ipynb)
- Edit your name in the cell above - your name should be the same as the filename
- Clean up Colab Notebooks folder
- Submit shared link

**Grading:** Pass/Fail (Complete all sections with justification)

### ✓ Part 1: Scenario

You have passed the initial phone screen for a Data Scientist role at "**OmniCorp,**" a mid-sized tech company. They have sent you this Jupyter Notebook as a take-home assessment.

**Your Instructions from the Hiring Manager:**

"We've attached some synthetic data representing two of our core business problems: estimating server costs (Regression) and predicting customer churn (Classification).

We don't care about hitting 99% accuracy. We care about **how you think**.

1. Check your assumptions.
2. Tell us what the coefficients *mean*.
3. Make a business recommendation based on your model."

## Instructions

1. Run the "Data Generation" cells to create your dataset.
2. Complete the **TODO** sections.
3. Answer the **Interview Questions** in the Markdown cells provided.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_squared_error, classification_report, confusion_matrix

# Setup
np.random.seed(101)
sns.set_style("whitegrid")
```

```
%matplotlib inline
```

## Task 1: Regression (Server Cost Estimation)

**Context:** OmniCorp wants to predict the hourly cost of running their cloud servers based on three features:

1. `CPU_Usage` (Percentage 0-100)
2. `Memory_Usage` (GB)
3. `Server_Age` (Years)

**The Catch:** The relationship isn't perfectly linear. Old servers become exponentially more expensive to maintain.

```
# --- DATA GENERATION (DO NOT MODIFY) ---
n = 1000
cpu = np.random.uniform(10, 90, n)
memory = np.random.uniform(4, 64, n)
age = np.random.uniform(0, 10, n)

# True relationship: Cost = Base + 2*CPU + 5*Memory +
# This exponential term on 'age' is the trap for a st
cost = 50 + 2 * cpu + 5 * memory + np.exp(0.5 * age)

df_servers = pd.DataFrame({'CPU': cpu, 'Memory': memo

# Split Data
X_reg = df_servers[['CPU', 'Memory', 'Age']]
y_reg = df_servers['Cost']
```

```
X_train_r, X_test_r, y_train_r, y_test_r = train_test_
df_servers.head()
```

```
# 🖥️ TODO: Fit a Standard Linear Regression Model (OLS)
# 1. Instantiate and fit LinearRegression to X_train_r
# 2. Predict on X_test_r
# 3. Calculate and print the RMSE (Root Mean Squared Error)

# [YOUR CODE HERE]

# 🖥️ TODO: Residual Plot
# Plot the Predicted Values (x-axis) vs. Residuals (A vertical column)
# Look for a pattern. If it's random cloud, good. If funnel shape, bad.

# [YOUR CODE HERE]
```



## Interview Question 1

**The Hiring Manager asks:** "We noticed your Residual Plot has a distinct 'funnel' or 'U' shape. What does that imply about the assumptions of Linear Regression? How would you fix this model to better capture the server age dynamics?"

**Your Answer:**

- **Observation:** [Write here]
- **Violation:** [Which OLS assumption is violated?  
Homoscedasticity? Linearity?]
- **Fix:** [Would you use a Log Transform? A Polynomial feature? A Gamma GLM? Pick one and explain why.]



## Task 2: Classification (Churn

### Prediction)

**Context:** The marketing team wants to predict which customers will cancel their subscription (`Churn = 1`) next month. **The Catch:** The data is **imbalanced**. Most customers are happy (0); only a few churn (1).

```
# --- DATA GENERATION (DO NOT MODIFY) ---
from sklearn.datasets import make_classification

# Generate Imbalanced Data (90% Class 0, 10% Class 1)
X_cls, y_cls = make_classification(n_samples=1000, n_
                                    n_redundant=1, n_c
                                    flip_y=0.01, rando

# Convert to DataFrame for readability
cols = ['Avg_Login_Time', 'Support_Tickets', 'Bill_Am
df_churn = pd.DataFrame(X_cls, columns=cols)
df_churn['Churn'] = y_cls

X_train_c, X_test_c, y_train_c, y_test_c = train_test
```

```
# 🖥 TODO: Fit a Logistic Regression Model
# 1. Fit LogisticRegression to the training data.
# 2. Print the Accuracy Score.

# [YOUR CODE HERE]
```

```
# 🖥️ TODO: Print the Confusion Matrix  
# Use sklearn.metrics.confusion_matrix  
  
# [YOUR CODE HERE]
```



## Interview Question 2

**The Hiring Manager asks:** "You reported an Accuracy of 90%+. That sounds amazing! But looking at the Confusion Matrix, how many *actual*/churners did we catch? Why is 'Accuracy' a dangerous metric for this specific dataset?"

**Your Answer:**

- [Discuss the False Negatives vs False Positives]
- [Explain why Accuracy is misleading in imbalanced datasets]



## Task 3: The Business Trade-off (Thresholding)

**Context:**

- A marketing campaign to retain a user costs **\$10**.
- If we fail to identify a churner, we lose their **\$100** lifetime value.

- **False Positive Cost:** \$10 (Wasted marketing on a happy user).
- **False Negative Cost:** \$100 (Lost customer).

We cannot treat these errors equally. We need to adjust our **Decision Threshold**.

```
# 🖥️ TODO: Threshold Moving
# 1. Get the predicted probabilities using .predict_proba
# 2. Instead of the default 0.5 threshold, try a threshold of 0.2
# 3. Calculate the "Business Cost" for both thresholds

# HINT:
# Cost = (False Positives * 10) + (False Negatives * 100)

model_log = LogisticRegression()
model_log.fit(X_train_c, y_train_c)
probs = model_log.predict_proba(X_test_c)[:, 1]

# Default Threshold (0.5)
preds_default = (probs > 0.5).astype(int)
tn, fp, fn, tp = confusion_matrix(y_test_c, preds_default)
cost_default = (fp * 10) + (fn * 100)

# Aggressive Threshold (0.2)
preds_aggressive = (probs > 0.2).astype(int)
tn_a, fp_a, fn_a, tp_a = confusion_matrix(y_test_c, preds_aggressive)
cost_aggressive = (fp_a * 10) + (fn_a * 100)

print(f"Cost at Threshold 0.5: ${cost_default}")
print(f"Cost at Threshold 0.2: ${cost_aggressive}")
```



## Interview Question 3

**The Hiring Manager asks:** "Based on the costs calculated above, which model deployment strategy do you recommend? Should we spam more people (Low Threshold) or be conservative (High Threshold)? Why?"

**Your Answer:**

- [Recommendation]
- [Justification based on ROI]



## Task 4: Random Forest & Feature Importance

**Context:** The engineering team wants to know *which* features actually drive churn so they can fix the product. Logistic Regression coefficients can be hard to interpret if data isn't scaled. Let's use a Random Forest.

```
# 🖥️ TODO: Fit a Random Forest & Plot Feature Importances
# 1. Fit a RandomForestClassifier to the churn data.
# 2. Extract `feature_importances_`
# 3. Create a bar plot showing which feature is most important

# [YOUR CODE HERE]
```



## Interview Question 4

**The Hiring Manager asks:** "Your Random Forest performs slightly better than the Logistic Regression, but it's a 'Black Box.' If the CEO asks *how* 'Support\_Tickets' impacts churn (e.g., does more tickets = more churn?), how would you derive that directionality from a Random Forest?"

**Your Answer:**

- [HINT: Feature Importance tells you *magnitude* but not *direction*. How do we find direction? Partial Dependence Plots? SHAP values? Correlation matrix check?]



## Part 2: Scenario

**OmniCorp Marketing Dept.** has a problem: They are treating every customer the same. They send the same "10% Off" email to everyone, from the college student buying socks to the enterprise client buying servers.

**Your Instructions from the Head of Marketing:**

"We have a dataset of user behavior on our platform. We don't know who these people are (no labels), but we suspect there are at least 3 distinct groups.

1. Find the groups (Clusters).
2. Tell us what defines them (e.g., 'Big Spenders').
3. We also suspect there are some 'Bots' messing up our analytics. Can you identify them?"

## Instructions

1. Run the "Data Generation" cells.
2. Complete the **TODO** sections.
3. Answer the **Interview Questions**.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, DBSCAN
from sklearn.metrics import silhouette_score

# Setup
np.random.seed(42)
sns.set_style("whitegrid")
%matplotlib inline
```



## Data Generation (Do Not Modify)

We are generating data with **5 Features**:

1. `Avg_Session_Time`: How long they stay on the site (minutes).
2. `Items_Viewed`: Number of distinct products looked at.
3. `Cart_Value`: Total value of items in cart (\$).
4. `Return_Rate`: Percentage of items returned (0.0 to 1.0).
5. `Support_Tickets`: Number of complaints filed.

```
from sklearn.datasets import make_blobs

# We create 3 "Natural" Customer Segments + 1 "Bot" C
# Segment 1: Window Shoppers (High views, low cart, 1
# Segment 2: Big Spenders (High cart, high session, 1
# Segment 3: Complainants (Low cart, high tickets, hig
# Segment 4: Bots (Super fast, high views, 0 cart, 0

data_1 = np.random.normal(loc=[15, 20, 50, 0.1, 1], s
data_2 = np.random.normal(loc=[45, 10, 500, 0.05, 0],
data_3 = np.random.normal(loc=[10, 5, 30, 0.8, 5], sc
data_bots = np.random.normal(loc=[0.5, 100, 0, 0, 0],

# Combine
data = np.vstack([data_1, data_2, data_3, data_bots])
df_users = pd.DataFrame(data, columns=['Avg_Session_T

# Shuffle
df_users = df_users.sample(frac=1).reset_index(drop=T

print(f"Dataset Shape: {df_users.shape}")
df_users.head()
```



# Task 1: Preprocessing & PCA

**Context:** The features are on totally different scales.

`Cart_Value` is in the hundreds, `Return_Rate` is 0-1. If you run K-Means now, `Cart_Value` will dominate the distance metric.

**Interview Logic:** You can't visualize 5 dimensions. We need to squash it down to 2 dimensions to "see" the clusters.

```
# 🖥 TODO: Scaling & PCA  
# 1. Initialize a StandardScaler and transform df_use  
# 2. Initialize PCA with n_components=2. Fit and tran  
# 3. Create a Scatter Plot of PC1 vs PC2.  
  
# [YOUR CODE HERE]
```



## Interview Question 1

**The Hiring Manager asks:** "I see you used PCA to visualize the data. Look at the variance explained ratio (print it if you haven't). Did we lose a lot of information by compressing 5 dimensions into 2? If we lost 40% of the variance, is this plot still trustworthy?"

**Your Answer:**

- [Discuss Explained Variance Ratio]
- [Is the tradeoff worth it for visualization?]

## ✓ Task 2: K-Means Clustering

**Context:** We need to group these users. We don't know  $K$ .

```
# 🖥️ TODO: Find Optimal K
# 1. Run K-Means for k=2 to k=10.
# 2. Calculate the Inertia (WCSS) for each k.
# 3. Plot the Elbow Curve.
# 4. (Optional but Recommended) Calculate Silhouette

# [YOUR CODE HERE]

# 🖥️ TODO: Apply Final K-Means
# 1. Choose your "Best K" (Hint: Look for the elbow o
# 2. Fit K-Means to X_scaled.
# 3. Add the 'Cluster_Labels' to your original DataFr

# [YOUR CODE HERE]
```

## ✓ Task 3: Cluster Interpretation (The "Business" Part)

**Context:** Getting "Cluster 0, 1, 2" means nothing to the Marketing Director. You need to name them.

```
# 🖥️ TODO: Profiling
# Group by 'Cluster_Labels' and calculate the MEAN of
# Look at the differences.
```

```
# print(df_users.groupby('Cluster_Labels').mean())
```



## Interview Question 2

**The Hiring Manager asks:** "Okay, you found 4 clusters. Based on the means you calculated above, give each cluster a **Marketing Persona Name** (e.g., 'Loyalists', 'Window Shoppers') and explain why."

**Your Answer:**

- **Cluster 0:** Name = [Name] | Reason: [High cart value? Low time?]
- **Cluster 1:** Name = [Name] | Reason: ...
- **Cluster 2:** Name = [Name] | Reason: ...
- **Cluster 3:** Name = [Name] | Reason: ...



## Task 4: Anomaly Detection (DBSCAN)

**Context:** Your K-Means model likely forced the "Bots" into a cluster, or maybe merged them with another group. Bots are weird. They have extreme values (e.g., 100 items viewed in 0.5 minutes). They are "density" outliers compared to normal human behavior.

Let's see if DBSCAN can isolate them better as "Noise" or a distinct, tight cluster.

```
# 🖥 TODO: DBSCAN  
# 1. Fit DBSCAN to X_scaled.  
# Note: You will need to tune 'eps' and 'min_samples'  
# Try eps=0.5, min_samples=5 as a starting point.  
# 2. Plot PC1 vs PC2 again, but color by DBSCAN label  
# (Remember: DBSCAN labels noise as -1).  
  
# [YOUR CODE HERE]
```



## Interview Question 3

**The Hiring Manager asks:** "In K-Means, every point *must* belong to a cluster. In DBSCAN, some points are labeled -1.

1. What does -1 represent in this specific dataset?
2. If you were building a real-time fraud detection system, would you use K-Means or DBSCAN? Why?"

**Your Answer:**

- [Explain Noise/Outliers]
- [Compare K-Means (Centroid based) vs DBSCAN (Density based) for fraud]

## Part 3: Scenario

"**GreenGrid Solutions**" has hired you to optimize a large-scale battery storage system.

- **The Goal:** Buy electricity when it's cheap, store it, and sell it back to the grid when it's expensive.
- **The Constraint:** The battery has limited capacity, and charging/discharging degrades it slightly (cost).
- **The Problem:** You don't have a labeled dataset of "Correct Actions." You only have the market environment.

## Your Instructions from the CTO:

"We need an RL Agent that learns to trade energy.

1. Build a custom OpenAI Gym-style environment.
2. Train a Q-Learning agent to maximize profit.
3. Explain how you would deploy this if customer data privacy was a concern (Federated Learning)."

## Instructions

1. Run the Environment Setup.
2. Implement the **Q-Learning Algorithm**.
3. Answer the **System Design Questions**.

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

```
# Setup
np.random.seed(42)
sns.set_style("whitegrid")
%matplotlib inline
```

## ▼ Task 1: The Environment (GridWorld for Batteries)

We will define a simple class `BatteryEnv`.

- **State:** Current Battery Level (0% to 100%, discretized into 10 steps).
- **Action:** 0=Hold, 1=Charge (Buy), 2=Discharge (Sell).
- **Reward:** (Price\_Sold - Price\_Bought) - Cost\_Degradation.

```
# --- SIMULATED MARKET DATA (DO NOT MODIFY) ---
# 24-hour cycle of electricity prices (Low at night,
prices = np.array([10, 10, 10, 12, 15, 20, 30, 45, 60
                   30, 25, 20, 18, 25, 40, 80, 100, 90,
                   80, 70, 60, 50, 45, 40, 35, 30])
# Plot Prices
plt.figure(figsize=(10, 4))
plt.plot(prices, marker='o', linestyle='--')
plt.title("24-Hour Electricity Price Cycle")
plt.xlabel("Hour")
plt.ylabel("Price ($/MWh)")
plt.show()

# --- ENVIRONMENT CLASS ---
```

```
class BatteryEnv:  
    def __init__(self, prices):  
        self.prices = prices  
        self.max_capacity = 10  
        self.current_charge = 0  
        self.time_step = 0  
        self.end_step = len(prices) - 1  
  
    def reset(self):  
        self.current_charge = 0  
        self.time_step = 0  
        return (self.current_charge, self.time_step)  
  
    def step(self, action):  
        # Current Price  
        current_price = self.prices[self.time_step]  
        reward = 0  
  
        # Execute Action  
        if action == 1: # CHARGE (Buy)  
            if self.current_charge < self.max_capacity:  
                self.current_charge += 1  
                reward = -current_price # Cost money  
            else:  
                reward = -50 # Penalty for trying to  
  
        elif action == 2: # DISCHARGE (Sell)  
            if self.current_charge > 0:  
                self.current_charge -= 1  
                reward = current_price # Gain money  
            else:  
                reward = -50 # Penalty for trying to  
  
        # Action 0 (Hold) has reward 0  
  
        # Move Time Forward  
        self.time_step += 1
```

```
done = (self.time_step >= self.end_step)
next_state = (self.current_charge, self.time_
              

return next_state, reward, done
```

## ▼ Task 2: Build the Agent (Q-Learning)

**Context:** You need to fill in the Q-Learning update loop.

Recall the Bellman Equation:

$$Q(s, a) = Q(s, a) + \alpha[R + \gamma \max Q(s', a') - Q(s, a)]$$

```
# 💻 TODO: Implement Q-Learning Training Loop

# Hyperparameters
alpha = 0.1    # Learning Rate
gamma = 0.95   # Discount Factor (Care about future pr
epsilon = 0.1  # Exploration Rate
episodes = 1000

# Initialize Q-Table: [Charge_Level(11), Time_Step(24
q_table = np.zeros((11, 24, 3))

env = BatteryEnv(prices)

rewards_history = []

for ep in range(episodes):
    state = env.reset() # Returns (charge, time)
    done = False
    total_reward = 0

    while not done:
        charge, time = state
```

```

# 1. Epsilon-Greedy Action Selection
if np.random.uniform(0, 1) < epsilon:
    action = np.random.choice([0, 1, 2]) # Ex
else:
    action = np.argmax(q_table[charge, time])

# 2. Take Action
next_state, reward, done = env.step(action)
next_charge, next_time = next_state

# 3. Update Q-Table (THE TODO PART)
# [YOUR CODE HERE]
# Implement the line: old_value + alpha * (re

# q_table[charge, time, action] = ...

state = next_state
total_reward += reward

rewards_history.append(total_reward)

# Plot Learning Curve
plt.plot(rewards_history)
plt.title("Agent Profit over Episodes")
plt.xlabel("Episode")
plt.ylabel("Total Profit")
plt.show()

```



## Interview Question 1

**The CTO asks:** "Look at your training curve. It likely goes up and then flattens out.

1. Why does it fluctuate so much even after 'learning'?  
(Hint: Look at `epsilon`).
2. If we deployed this to a real grid where prices spike unpredictably (unlike this fixed 24h cycle), how would you modify the state definition to handle that?"

### Your Answer:

- [Discuss Exploration vs Exploitation noise]
- [Discuss State Space augmentation (e.g., adding 'Current\_Price' or 'Forecast\_Price' to the state tuple)]

## 🤝 Task 3: Transfer & Federated Learning (Design)

**Context:** GreenGrid wants to install these batteries in 10,000 residential homes.

1. **Issue A:** Training a model from scratch for *every single house* takes too long (2 weeks per house).
2. **Issue B:** Homeowners are worried that GreenGrid will spy on their energy usage patterns (privacy).

## 🎙 Interview Question 2 (Transfer Learning)

**The CTO asks:** "We have a perfectly trained model for House A (Solar Panels, Texas weather). House B (Wind Turbine, Chicago weather) comes online tomorrow. Instead of random

initialization, how can we use **Transfer Learning** to speed up House B's training? Which parts of the Q-Network (if we used Deep RL) would you freeze vs. retrain?"

### **Your Answer:**

- [Concept: Warm Start]
- [Strategy: Transfer general grid dynamics, fine-tune for local weather patterns]



### Interview Question 3 (Federated Learning)

**The CTO asks:** "Explain to the Legal Team how **Federated Learning** solves the privacy issue. Specifically, describe what information is sent to the cloud and what stays on the device."

### **Your Answer:**

- [Data stays local]
- [Only Weight Updates (Gradients) are shared]
- [Aggregation (FedAvg) happens in the cloud]

**Remember:** In the interview, your code must run, but your **explanations** get you the job.

