Question 1

Logistic Regression is a statistical model used for binary and multiclass classification problems. It predicts the probability of a class by mapping a linear combination of input features through a Sigmoid function, which outputs values between 0 and 1.

Differences from Linear Regression: - Purpose: Classification vs Regression - Function: Sigmoid vs Linear - Output Range: (0,1) vs (-∞,+∞) - Error Metric: Log Loss vs MSE

Question 2

The Sigmoid function maps the linear output of the model into a probability between 0 and 1. Formula: σ(z) = 1 / (1 + e^-z). It allows threshold-based classification decisions.

Question 3

Regularization prevents overfitting by adding a penalty to large coefficients. Types: - L1 (Lasso): Absolute value penalty - L2 (Ridge): Squared penalty

Question 4

Common metrics: - Accuracy - Precision - Recall - F1-Score - ROC-AUC They give a complete view of model performance, especially with imbalanced data.

Question 5

Python Code:

```
import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression


# Step 1: Load dataset from sklearn

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target


# Step 2: Save to CSV (simulating a CSV file)

csv_filename = "iris_dataset.csv"

df.to_csv(csv_filename, index=False)
```

```
# Step 3: Load CSV into DataFrame
data = pd.read_csv(csv_filename)

# Step 4: Split into features (X) and target (y)
X = data.drop('target', axis=1)
y = data['target']

# Step 5: Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Step 6: Train Logistic Regression model
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)

# Step 7: Print accuracy
accuracy = model.score(X_test, y_test)
print(f"Accuracy: {accuracy:.2f}")
```

Question 6

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Load dataset from sklearn
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Logistic Regression with L2 regularization (default penalty)
model = LogisticRegression(penalty='l2', solver='lbfgs', max_iter=200)
model.fit(X_train, y_train)

# Print coefficients
print("Model Coefficients (per class):")
for class_index, coef in enumerate(model.coef_):
    print(f"Class {class_index}: {coef}")

# Accuracy
accuracy = model.score(X_test, y_test)
print(f"\nAccuracy: {accuracy:.2f}")
```

Question 7

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# Load dataset from sklearn
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Logistic Regression for multiclass using one-vs-rest
model = LogisticRegression(
    multi_class="ovr",      # one-vs-rest
    solver="lbfgs",         # supports ovr
    max_iter=500,
)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)
```

```
# Classification report
print("Classification Report (One-vs-Rest):")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

Question 8

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression

# Load dataset
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Define parameter grid
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear']  # liblinear supports both l1 and l2 penalties
}
```

```
# Logistic Regression
log_reg = LogisticRegression(max_iter=500)

# GridSearchCV
grid_search = GridSearchCV(
    estimator=log_reg,
    param_grid=param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)

grid_search.fit(X_train, y_train)

# Best parameters and score
print("Best Parameters:", grid_search.best_params_)
print(f"Best Cross-Validation Accuracy: {grid_search.best_score_:.2f}")

# Test set accuracy
test_accuracy = grid_search.score(X_test, y_test)
print(f"Test Set Accuracy: {test_accuracy:.2f}")
```

```
Best Parameters: {'C': 10, 'penalty': 'l1', 'solver': 'liblinear'}
Best Cross-Validation Accuracy: 0.97
Test Set Accuracy: 0.97
```

Question 9

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

# Load dataset
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Logistic Regression without scaling
model_no_scale = LogisticRegression(max_iter=500)
model_no_scale.fit(X_train, y_train)
acc_no_scale = model_no_scale.score(X_test, y_test)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Logistic Regression with scaling
model_scaled = LogisticRegression(max_iter=500)
model_scaled.fit(X_train_scaled, y_train)
acc_scaled = model_scaled.score(X_test_scaled, y_test)

# Compare results
print(f"Accuracy without scaling: {acc_no_scale:.2f}")
print(f"Accuracy with scaling:    {acc_scaled:.2f}")
```

```
Accuracy without scaling: 0.97
Accuracy with scaling:    0.93
```

Question 10

Logistic Regression model for an imbalanced marketing response prediction problem step-by-step, in a way that's realistic for an e-commerce business scenario.

1. Understanding the Business Context

- Goal: Predict which customers are most likely to respond to a marketing campaign.
- Data: Customer demographics, transaction history, browsing patterns, loyalty points.
- Challenge: Only ~5% response rate, indicating strong class imbalance.

2. Data Preparation

a) Data Cleaning

- Handle missing values (imputation or removal).
- Remove duplicates.
- Ensure correct data types.

b) Feature Engineering

- Create RFM features (Recency, Frequency, Monetary value).
- Aggregate purchase behavior metrics.
- Encode categorical variables (One-Hot or Target Encoding).
- Extract time-based features.

3. Handling Class Imbalance

- Use class_weight='balanced' in Logistic Regression.
- Apply SMOTE to oversample positive cases.

- Try undersampling majority class cautiously.
- Start with class_weight='balanced' for simplicity.

4. Feature Scaling

- Logistic Regression benefits from scaling for faster convergence.
- Use :

StandardScaler after splitting train/test sets.

5. Model Training

- Use:

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(
    penalty='l2',
    solver='liblinear',
    class_weight='balanced',
    max_iter=500
)
```

6. Hyperparameter Tuning

Use GridSearchCV:

- Tune **C** (inverse of regularization strength) — e.g., [0.01, 0.1, 1, 10].
- Try different **penalties** (l1, l2).
- Possibly test different solvers compatible with penalties.

```
param_grid = {
  'C': [0.01, 0.1, 1, 10],
  'penalty': ['l1', 'l2'],
  'solver': ['liblinear']
}
```

7. Model Evaluation

- Accuracy is not reliable for imbalanced data.
- Use precision, recall, F1-score, ROC-AUC, PR-AUC.
- Plot ROC and Precision-Recall curves.
- Analyze confusion matrix for FP/FN trade-offs.

8. Deployment & Business Use

- Output probabilities for ranking customers.
- Adjust classification threshold based on business goals.
- Integrate into campaign automation systems.