```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from scipy.special import linestyle
from sklearn.metrics import confusion_matrix,  accuracy_score, precision_score,
recall_score

#%%-----------------Q1--------------------
data = pd.DataFrame({
    'Customer#':[1, 2, 3, 4, 5, 6],
    'Income': [60, 64.8, 84, 59, 108, 75],
    'Lot_size':[18.4, 21.6, 17.6, 16, 17.6, 19.6],
    'Ownership':['Owner','Owner','Nonowner','Nonowner','Owner','Nonowner']
})

beta_0 = -25.9382
beta_1 = 0.1109
beta_2 = 0.9638

data['Logit_p'] = beta_0 + beta_1 * data['Income'] + beta_2 * data['Lot_size']
print("Logistic regression model:")
print(f"Logit(p) = {beta_0:.4f} + {beta_1:.4f} * Income + {beta_2:.4f} * Lot size")

data['Probability'] = 1/(1 + np.exp(-data['Logit_p']))
data['Predicted_Ownership'] = np.where(data['Probability']>=0.75, 'Owner',
'Nonowner')

print("\nCustomer classification based on probability cutoff = 0.75 using logistic
regression model:")
for index,row in data.iterrows():
    print(f"Customer {int(row['Customer#'])} is classified as
{row['Predicted_Ownership']}")

y_true = data['Ownership'].apply(lambda x: 1 if x == 'Owner' else 0)
y_pred = data['Predicted_Ownership'].apply(lambda x: 1 if x == 'Owner' else 0)
matrix = confusion_matrix(y_true, y_pred)
print("\nConfusion matrix:")
print(matrix)

TP = matrix[1,1]
TN = matrix[0,0]
FP = matrix[0,1]
FN = matrix[1,0]

accuracy = (TP+TN)/(TP+TN+FP+FN)
precision = TP /(TP+FP) if (TP+FP)!=0 else 0
recall = TP/(TP+FN) if (TP+FN)!=0 else 0

print("\nClassfification results:")
print(f"Accuracy: {accuracy:.3f}")
print(f"Precision: {precision:.3f}")
print(f"Recall: {recall:.3f}")

#%%-----------------Q2--------------------
import numpy as np
import matplotlib.pyplot as plt
z = np.linspace(-10, 10, 1000)
sigma_z = 1 / (1 + np.exp(-z))
loss_y1 = -np.log(sigma_z)
```

```
loss_y0 = -np.log(1 - sigma_z)
plt.figure(figsize=(10, 6))
plt.plot(sigma_z, loss_y1, label='J(w) if y=1', linewidth=3, color='darkblue')
plt.plot(sigma_z, loss_y0, label='J(w) if y=0', linewidth=3, color='darkorange',
linestyle='--')
plt.title('Log-Loss Function')
plt.xlabel('σ(z)')
plt.ylabel('J(w)')
plt.xlim(0, 1)
plt.ylim(0, 5)
plt.grid(True)
plt.legend()
plt.show()

#%%-----------------Q3--------------------
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import ConfusionMatrixDisplay, accuracy_score,
precision_score, recall_score
X,y = make_classification(
    n_samples=1000,
    n_features=2,
    n_clusters_per_class=2,
    n_informative=2,
    n_repeated=0,
    n_redundant=0,
    random_state=0
)
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,
random_state=5805,shuffle=True,stratify=y)
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.fit_transform(X_test)
model_logistic = LogisticRegression(random_state=5805)
model_logistic.fit(X_train,y_train)
y_pred_logistic = model_logistic.predict(X_test)
y_pred_logistic_probability = model_logistic.predict_proba(X_test)[:,1]
cnf_matrix = metrics.confusion_matrix(y_test,y_pred_logistic)
print("\nConfusion matrix:")
print(cnf_matrix)
disp = ConfusionMatrixDisplay(confusion_matrix=cnf_matrix, display_labels=['Class
0', 'Class 1'])
disp.plot(cmap='Blues')
plt.title('Confusion matrix')
plt.show()

logistic_fpr, logistic_tpr,_ =
metrics.roc_curve(y_test,y_pred_logistic_probability)
auc_logistic = metrics.roc_auc_score(y_test,y_pred_logistic_probability)
plt.figure(figsize=(10,8))
plt.plot(logistic_fpr,logistic_tpr,label=f'Logistic Regression AUC
{auc_logistic:.3f} curve')
plt.plot(logistic_fpr,logistic_fpr,color='red',linestyle='--')
plt.title('ROC Curve for Logistic Regression Model')
```

```
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')

plt.legend(loc=4)
plt.grid()
plt.show()

TP = cnf_matrix[1,1]
TN = cnf_matrix[0,0]
FP = cnf_matrix[0,1]
FN = cnf_matrix[1,0]

accuracy = (TP+TN)/(TP+TN+FP+FN)
precision = TP /(TP+FP) if (TP+FP)!=0 else 0
recall = TP/(TP+FN) if (TP+FN)!=0 else 0

print("\nClassfification results:")
print(f"Accuracy: {accuracy:.3f}")
print(f"Precision: {precision:.3f}")
print(f"Recall: {recall:.3f}")

#%%-----------------Q4.a--------------------
import pandas as pd
from imblearn.over_sampling import SMOTE
url ='https://raw.githubusercontent.com/rjafari979/Information-Visualization-Data-
Analytics-Dataset-/refs/heads/main/Smarket.csv'
df = pd.read_csv(url)
print("Initial distribution of 'Direction':")
print(df['Direction'].value_counts())
df['Direction'].value_counts().plot(kind='bar', title='Imbalanced Dataset')
plt.xticks(rotation=0)
plt.show()
smote = SMOTE(random_state=5805)
X = df[['Lag1', 'Lag2', 'Lag3', 'Lag4', 'Lag5', 'Volume']]
y = df['Direction'].apply(lambda x: 1 if x == 'Up' else 0)
X_resampled, y_resampled = smote.fit_resample(X, y)
y_resampled_labels = pd.Series(y_resampled).map({1: 'Up', 0: 'Down'})
pd.Series(y_resampled_labels).value_counts().plot(kind='bar', title='Balanced
Dataset after SMOTE')
plt.xticks(rotation=0)
plt.show()
print("Final distribution of 'Direction' after SMOTE:")
print(y_resampled_labels.value_counts())

#%%-----------------Q4.b--------------------
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X_resampled,y_resampled,test_size=0.2,
random_state=5805,shuffle=True)
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
X_train = pd.DataFrame(X_train, columns=X.columns)
X_test = pd.DataFrame(X_test, columns=X.columns)
print("\nPrinting first 5 rows of the train dataset...")
print(X_train.head())
print("\nPrinting first 5 rows of the test dataset...")
print(X_test.head())
```

```
#%%-----------------Q4.c--------------------
model = LogisticRegression(random_state=5805)
model.fit(X_train, y_train)
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
y_test_pred_prob = model.predict_proba(X_test)[:, 1]
train_score = model.score(X_train, y_train)
test_score = model.score(X_test, y_test)
print(f"\nTrain score: {train_score:.3f}")
print(f"\nTest score: {test_score:.3f}")
matrix_confusion = metrics.confusion_matrix(y_test,y_test_pred)
print("\nConfusion matrix:")
print(matrix_confusion)
disp = ConfusionMatrixDisplay(confusion_matrix=matrix_confusion,
display_labels=['Down', 'Up'])
disp.plot(cmap='Blues')
plt.title('Confusion matrix')
plt.show()

fpr, tpr,_ = metrics.roc_curve(y_test,y_test_pred_prob)
auc = metrics.roc_auc_score(y_test,y_test_pred_prob)
plt.figure(figsize=(10,8))
plt.plot(fpr,tpr,label=f'Logistic Regression AUC {auc:.3f} curve')
plt.plot(fpr,fpr,color='red',linestyle='--')
plt.title('ROC Curve for Logistic Regression Model')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc=4)
plt.grid()
plt.show()

TP = matrix_confusion[1,1]
TN = matrix_confusion[0,0]
FP = matrix_confusion[0,1]
FN = matrix_confusion[1,0]

accuracy = (TP+TN)/(TP+TN+FP+FN)
precision = TP /(TP+FP) if (TP+FP)!=0 else 0
recall = TP/(TP+FN) if (TP+FN)!=0 else 0

print("\nClassfification results:")
print(f"Accuracy: {accuracy:.3f}")
print(f"Precision: {precision:.3f}")
print(f"Recall: {recall:.3f}")

#%%-----------------Q4.d--------------------
from sklearn.model_selection import  GridSearchCV
from sklearn.metrics import f1_score
param_grid = {
    'penalty': ['l1', 'l2', 'elasticnet'],
    'l1_ratio': np.linspace(0, 1, 30),
    'C': np.logspace(-3, 1, 10)
}

grid_search = GridSearchCV(
    LogisticRegression(solver='saga', random_state=5805, max_iter=10000),
    param_grid,
    cv=5,
    scoring='accuracy',
```

```
    n_jobs=-1
)

grid_search.fit(X_train, y_train)

# Display the best parameters
print("\nBest parameters from Grid Search with CV:")
print(grid_search.best_params_)

# Fit and evaluate the model using the best parameters
best_model = grid_search.best_estimator_
y_test_pred_best = best_model.predict(X_test)
y_test_pred_prob_best = best_model.predict_proba(X_test)[:, 1]

# Print the best scores for the train and test using the tuned model
train_score_best = best_model.score(X_train, y_train)
test_score_best = best_model.score(X_test, y_test)
print(f"\nTrain score (tuned model):, {train_score_best:.3f}")
print(f"Test score (tuned model):, {test_score_best:.3f}")

# Plot the confusion matrix for the tuned model
cm_best = metrics.confusion_matrix(y_test, y_test_pred_best)
disp_best = ConfusionMatrixDisplay(confusion_matrix=cm_best,
display_labels=['Down', 'Up'])
disp_best.plot(cmap='Blues')
plt.title('Confusion Matrix (Tuned Model)')
plt.show()

# Plot the ROC curve with AUC for the tuned model
fpr_best, tpr_best, _ = metrics.roc_curve(y_test, y_test_pred_prob_best)
roc_auc_best = metrics.roc_auc_score(fpr_best, tpr_best)

plt.figure(figsize=(10, 6))
plt.plot(fpr_best, tpr_best, color='green', lw=2, label=f'ROC curve (AUC =
{roc_auc_best:.2f})')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve (Tuned Model)')
plt.legend(loc='lower right')
plt.grid()
plt.show()

# Display the accuracy, precision, recall, and F1 score for the tuned model
accuracy_best = accuracy_score(y_test, y_test_pred_best)
precision_best = precision_score(y_test, y_test_pred_best)
recall_best = recall_score(y_test, y_test_pred_best)
f1_best = f1_score(y_test, y_test_pred_best)

print("\nClassification Metrics (Tuned Model):")
print(f"Accuracy: {accuracy_best:.2f}")
print(f"Precision: {precision_best:.2f}")
print(f"Recall: {recall_best:.2f}")
print(f"F1 Score: {f1_best:.2f}")

# Check if the grid search improved performance
improvement = test_score_best > test_score
print("\nDid the grid search improve the performance?", improvement)
```