ECON 5140: Applied Econometrics

Homework 1: GLMs & Time Series Foundations

PART 1: ANALYSIS PROBLEMS

Problem 1: Logistic Regression - Loan Default Prediction

A bank wants to predict loan defaults based on credit score. They have the following data from a logistic regression:

Model: log(p/(1-p)) = ?? + ??(CreditScore)

Estimated coefficients:

??? = 5.2

??? = -0.008

where p = P(Default = 1 | CreditScore)

Questions:

a) Calculate the predicted probability of default for someone with a credit score of 650.

b) Calculate the predicted probability of default for someone with a credit score of 750.

c) At what credit score is the predicted probability of default equal to 0.5? (This is the decision boundary)

d) Interpret the coefficient ??? = -0.008 in terms of odds ratios. What happens to the odds of default when credit score increases by 100 points?

Problem 2: Poisson Regression - Website Visits

An e-commerce company models daily website visits using Poisson regression.

Model: log(E[Visits|X]) = ?? + ??(Ad_Spend) + ??(Weekend)

where:

Ad_Spend is in thousands of dollars

Weekend = 1 if weekend, 0 otherwise

Estimated coefficients:

??? = 5.5

??? = 0.12

??? = 0.30

Questions:

a) Predict the expected number of visits on a weekday with $5,000 ad spend.

b) Interpret ??? = 0.12. What is the percentage change in expected visits for each additional $1,000 in ad spending?

c) Interpret ??? = 0.30. What is the percentage increase in visits on weekends compared to weekdays?

d) If the company wants to achieve 500 visits on a weekday, how much should they spend on ads?

Problem 3: Autocorrelation Calculation

A tech company tracks daily active users (in thousands). Here are 10 consecutive days of data:

Questions:

a) Calculate the mean (?) and standard deviation (?) of the series.

b) Calculate the lag-1 autocorrelation ?(1) = Corr(Y?, Y???):

Hint: Create two series: Y? to Y? and Y? to Y??, then calculate correlation

c) What does this autocorrelation value tell you about the persistence in daily active users?

d) Based on this autocorrelation, would knowing Day 10's value help forecast Day 11? Why?

Problem 4: Moving Averages

Daily website traffic (in thousands of visits):

Questions:

a) Calculate a 3-day centered moving average for Day 4:

MA?(4) = (Y? + Y? + Y?) / 3

b) Calculate a 3-day trailing moving average for Day 5:

MA?(5) = (Y? + Y? + Y?) / 3

c) For Day 4, calculate a weighted moving average with weights (0.25, 0.50, 0.25):

WMA(4) = 0.25×Y? + 0.50×Y? + 0.25×Y?

d) Why might the weighted average produce a smoother trend estimate?

PART 2: CODING PROBLEMS

Problem 5: Customer Purchase Prediction & Time Series Analysis

You will work with two datasets: customer purchase behavior (GLM) and e-commerce sales over time (Time Series).

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from scipy import stats

import statsmodels.api as sm

from statsmodels.discrete.discrete_model import Logit, Probit, Poisson

from statsmodels.tsa.seasonal import STL, seasonal_decompose

from statsmodels.graphics.tsaplots import plot_acf

import warnings
```

```python
warnings.filterwarnings('ignore')

# Set random seed for reproducibility
np.random.seed(42)

print("=" * 70)
print("ECON 5140 - HOMEWORK 1")
print("Part A: Generalized Linear Models")
print("Part B: Time Series Decomposition")
print("=" * 70)

# ============================================================================
# DATASET 1: CUSTOMER PURCHASE DATA (for GLM analysis)
# ============================================================================
print("\n" + "=" * 70)
print("DATASET 1: Customer Purchase Behavior")
print("=" * 70)

n_customers = 1000

# Generate customer features
age = np.random.normal(35, 10, n_customers)
income = np.random.normal(50, 15, n_customers)  # in thousands
time_on_site = np.random.gamma(2, 3, n_customers)  # in minutes

# True relationship (latent variable)
z = -3 + 0.05*age + 0.04*income + 0.15*time_on_site + np.random.normal(0, 1, n_customers)

# Generate binary outcome (Purchase: 1=Yes, 0=No)
purchase = (z > 0).astype(int)

# Create DataFrame
df_customers = pd.DataFrame({
    'Age': age,
    'Income': income,
    'TimeOnSite': time_on_site,
    'Purchase': purchase
})

print(f"Number of customers: {len(df_customers)}")
print(f"Purchase rate: {df_customers['Purchase'].mean():.2%}")
print(f"\nFirst 5 rows:")
print(df_customers.head())

# ============================================================================
# DATASET 2: E-COMMERCE SALES TIME SERIES
# ============================================================================
print("\n" + "=" * 70)
print("DATASET 2: E-commerce Daily Sales")
print("=" * 70)

# Create 2 years of daily data
dates = pd.date_range('2024-01-01', '2025-12-31', freq='D')
n_days = len(dates)
t = np.arange(n_days)

# Components
trend = 1000 + 2*t + 0.01*t**2
yearly_seasonal = 200 * np.sin(2*np.pi*t/365) + 150 * np.cos(2*np.pi*t/365)
weekly_seasonal = 100 * np.sin(2*np.pi*t/7)

# Special events
special_events = np.zeros(n_days)
for year in [2024, 2025]:
    # Black Friday
    bf_date = pd.Timestamp(f'{year}-11-24')
    bf_idx = (dates == bf_date)
    special_events[bf_idx] = 800

    # Christmas
    xmas_idx = (dates >= f'{year}-12-20') & (dates <= f'{year}-12-25')
    special_events[xmas_idx] = 400
```

```
# Random noise
noise = np.random.normal(0, 50, n_days)

# Combine components
sales = trend + yearly_seasonal + weekly_seasonal + special_events + noise
sales = np.maximum(sales, 0)

# Create DataFrame
df_sales = pd.DataFrame({
    'Date': dates,
    'Sales': sales,
    'DayOfWeek': dates.dayofweek,
    'Month': dates.month,
    'IsWeekend': dates.dayofweek >= 5
})
df_sales.set_index('Date', inplace=True)

print(f"Date range: {df_sales.index[0].date()} to {df_sales.index[-1].date()}")
print(f"Number of days: {len(df_sales)}")
print(f"\nSales Statistics:")
print(df_sales['Sales'].describe())

# ============================================================================
# PART A: GENERALIZED LINEAR MODELS
# ============================================================================
print("\n" + "=" * 70)
print("PART A: GENERALIZED LINEAR MODELS")
print("=" * 70)

# ----------------------------------------------------------------------------
# A1: Exploratory Data Analysis (GLM)
# ----------------------------------------------------------------------------
print("\n" + "-" * 70)
print("A1: Exploratory Data Analysis")
print("-" * 70)
```

```
# YOUR CODE:
# 1. Create box plots comparing Age, Income, and TimeOnSite
#    between purchasers and non-purchasers
#    - Use 3 subplots (1 row, 3 columns)
#
# 2. Calculate and print mean values for each group:
#    - Mean Age: Purchasers vs Non-purchasers
#    - Mean Income: Purchasers vs Non-purchasers
#    - Mean TimeOnSite: Purchasers vs Non-purchasers
#
# 3. Create a correlation matrix heatmap for the features

# ----------------------------------------------------------------------
# A2: Linear Probability Model (LPM)
# ----------------------------------------------------------------------
print("\n" + "-" * 70)
print("A2: Linear Probability Model")
print("-" * 70)

# YOUR CODE:
# 1. Fit OLS model: Purchase ~ Age + Income + TimeOnSite
#    - Add constant using sm.add_constant()
#    - Use sm.OLS()
#
# 2. Print regression summary
#
# 3. Calculate predicted probabilities
#    - Count how many predictions are outside [0, 1]
#    - Print the percentage of invalid predictions
#
# 4. Create histogram of predicted probabilities
#    - Mark the [0, 1] boundaries with vertical lines

# ----------------------------------------------------------------------
# A3: Logistic Regression
# ----------------------------------------------------------------------
print("\n" + "-" * 70)
print("A3: Logistic Regression")
print("-" * 70)

# YOUR CODE:
# 1. Fit logistic regression: sm.Logit()
#
# 2. Print summary and extract:
#    - Coefficients
#    - Odds ratios: np.exp(coefficients)
#    - p-values
#
# 3. Interpret each coefficient:
#    - Age: Effect on log-odds
#    - Income: Effect on log-odds
#    - TimeOnSite: Effect on log-odds
#
# 4. Calculate predicted probabilities
#    - Verify all are in [0, 1]
#    - Create histogram


# ----------------------------------------------------------------------
# A4: Prediction for New Customers
# ----------------------------------------------------------------------
print("\n" + "-" * 70)
print("A4: Predictions for New Customers")
print("-" * 70)

# New customers
new_customers = pd.DataFrame({
    'Age': [25, 35, 45, 55],
    'Income': [30, 50, 70, 90],
    'TimeOnSite': [2, 5, 8, 10]
})

# YOUR CODE:
# 1. Use your logistic model to predict purchase probability
#    for each new customer
#
# 2. Create a nice formatted table showing:
#    - Customer features
#    - Predicted probability
#    - Classification (Purchase = 1 if p > 0.5)
#
# 3. Which customer is most likely to purchase? Why?

# ======================================================================
# PART B: TIME SERIES ANALYSIS
# ======================================================================
print("\n" + "=" * 70)
print("PART B: TIME SERIES ANALYSIS")
print("=" * 70)

# ----------------------------------------------------------------------
# B1: Time Series Visualization
# ----------------------------------------------------------------------
print("\n" + "-" * 70)
print("B1: Time Series Visualization")
print("-" * 70)

# YOUR CODE:
# 1. Create time series plot of daily sales
#    - Full 2-year series
#    - Label axes appropriately
#
# 2. Create seasonal subseries plots:
#    - Box plot: Sales by day of week
#    - Box plot: Sales by month
```

```
#
# 3. Calculate and print:
#    - Mean sales by day of week
#    - Mean sales by month
#
# 4. What patterns do you observe?

# -----------------------------------------------------------------
# B2: Stationarity Assessment
# -----------------------------------------------------------------
print("\n" + "-" * 70)
print("B2: Stationarity Check")
print("-" * 70)

# YOUR CODE:
# 1. Create a plot with 3 subplots:
#    - Original series
#    - Rolling mean (30-day window)
#    - Rolling std (30-day window)
#
# 2. Is the series stationary? Justify based on:
#    - Does mean change over time?
#    - Does variance change over time?
#
# 3. Compare first 6 months vs last 6 months:
#    - Calculate mean and std for each period
#    - Print comparison

# -----------------------------------------------------------------
# B3: Autocorrelation Analysis
# -----------------------------------------------------------------
print("\n" + "-" * 70)
print("B3: Autocorrelation Function")
print("-" * 70)
```

```
# YOUR CODE:
# 1. Plot ACF for up to 60 lags
#    - Use plot_acf from statsmodels
#
# 2. Calculate specific autocorrelations manually:
#    - Lag 1 (yesterday)
#    - Lag 7 (last week)
#    - Lag 30 (last month)
#    Use: np.corrcoef(sales[:-lag], sales[lag:])[0,1]
#
# 3. Interpret:
#    - Do you see weekly patterns?
#    - How persistent is the autocorrelation?

# ----------------------------------------------------------------------
# B4: STL Decomposition
# ----------------------------------------------------------------------
print("\n" + "-" * 70)
print("B4: STL Decomposition")
print("-" * 70)

# YOUR CODE:
# 1. Apply STL decomposition with weekly seasonality:
#    stl = STL(df_sales['Sales'], seasonal=7, robust=True)
#    result = stl.fit()
#
# 2. Plot all four components:
#    - Observed
#    - Trend
#    - Seasonal (weekly pattern)
#    - Remainder
#
# 3. Analyze each component:
#    - What is the trend pattern?
#    - What is the weekly seasonal pattern?
#    - Are special events visible in remainder?
#

# -------------------------------------------------------------------
# B5: Remainder Diagnostics
# -------------------------------------------------------------------
print("\n" + "-" * 70)
print("B5: Remainder Analysis")
print("-" * 70)

# YOUR CODE:
# 1. Extract remainder from STL
#
# 2. Create diagnostic plots:
#    - Time series plot of remainder
#    - Histogram of remainder
#    - ACF of remainder
#
# 3. Statistical tests:
#    - Mean (should be ? 0)
#    - Standard deviation
#    - Check normality
#
# 4. Identify outliers:
#    - Find days where |remainder| > 3×std
#    - Print dates and investigate
#    - Are Black Friday and Christmas visible?
```