Predictive Analysis & Optimization - Practical Answers (Code)

This PDF contains runnable Python code answers for the practical questions you uploaded. Each question includes a code snippet (and brief comment) you can copy and run locally.

Q: Data collection (online, local, CSV)

```
# Data collection examples (pandas)
import pandas as pd

# 1) From local CSV
df_local = pd.read_csv('data/local_file.csv')  # replace with your path

# 2) From online CSV (public URL)
url = 'https://people.sc.fsu.edu/~jburkardt/data/csv/airtravel.csv'
df_online = pd.read_csv(url)

# 3) From local Excel
# df_excel = pd.read_excel('data/local_file.xlsx', sheet_name=0)

# 4) From a database (example using sqlite)
import sqlite3
conn = sqlite3.connect(':memory:')  # or 'mydb.sqlite'
# df_db = pd.read_sql_query('SELECT * FROM table_name', conn)

print(df_local.shape if 'df_local' in globals() else 'local CSV placeholder')
print('Online sample rows:', df_online.shape)
```

North-West Corner Method (Transportation IBFS)

```
def northwest_corner(supply, demand):
    i = j = 0
    m, n = len(supply), len(demand)
    supply = supply.copy()
    demand = demand.copy()
    solution = [[0]*n for _ in range(m)]
    while i < m and j < n:
        quantity = min(supply[i], demand[j])
        solution[i][j] = quantity
        supply[i] -= quantity
        demand[j] -= quantity
        if supply[i] == 0:
            i += 1
        if demand[j] == 0:
            j += 1
    return solution

# Example (from the problem)
supply = [17,12,16]  # O1,O2,O3
demand = [14,8,23]   # D1,D2,D3
sol = northwest_corner(supply, demand)
print('NW solution (matrix):')
for row in sol:
    print(row)
```

Least Cost Method (Transportation IBFS)

```
def least_cost_method(costs, supply, demand):
    m, n = len(costs), len(costs[0])
    supply = supply.copy()
    demand = demand.copy()
    solution = [[0]*n for _ in range(m)]
```

```
    # Create list of all cells sorted by cost
    cells = [(costs[i][j], i, j) for i in range(m) for j in range(n)]
    cells.sort(key=lambda x: x[0])
    for cost, i, j in cells:
        if supply[i] == 0 or demand[j] == 0:
            continue
        qty = min(supply[i], demand[j])
        solution[i][j] = qty
        supply[i] -= qty
        demand[j] -= qty
    return solution

# Example costs and capacities
costs = [[8,6,10],[9,12,13],[14,9,16]]
supply = [20,30,25]
demand = [30,25,20]
sol = least_cost_method(costs, supply, demand)
for row in sol: print(row)
```

Vogel's Approximation Method (VAM)

```
def vogel_approximation(costs, supply, demand):
    m, n = len(costs), len(costs[0])
    supply = supply.copy()
    demand = demand.copy()
    solution = [[0]*n for _ in range(m)]
    rows = set(range(m))
    cols = set(range(n))
    import heapq
    while rows and cols:
        # compute penalties
        row_penalty = {}
        for i in rows:
            vals = sorted([costs[i][j] for j in cols])
            row_penalty[i] = (vals[1]-vals[0]) if len(vals)>1 else vals[0]
        col_penalty = {}
        for j in cols:
            vals = sorted([costs[i][j] for i in rows])
            col_penalty[j] = (vals[1]-vals[0]) if len(vals)>1 else vals[0]
        # pick max penalty
        max_row = max(row_penalty.items(), key=lambda x:x[1]) if row_penalty else (None, -1)
        max_col = max(col_penalty.items(), key=lambda x:x[1]) if col_penalty else (None, -1)
        if max_row[1] >= max_col[1]:
            i = max_row[0]
            # choose min cost in that row
            j = min(cols, key=lambda c: costs[i][c])
        else:
            j = max_col[0]
            i = min(rows, key=lambda r: costs[r][j])
        qty = min(supply[i], demand[j])
        solution[i][j] = qty
        supply[i] -= qty
        demand[j] -= qty
        if supply[i] == 0: rows.remove(i)
        if demand[j] == 0: cols.remove(j)
    return solution

# Example
costs = [[21,16,15,13],[17,18,14,23],[32,27,18,41]]
supply = [11,13,19]
demand = [6,10,12,15]
sol = vogel_approximation(costs, supply, demand)
for row in sol: print(row)
```

Regression (example: linear regression)

```
# Simple linear regression example
import pandas as pd
```

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Sample synthetic data
import numpy as np
X = np.arange(100).reshape(-1,1)
y = 2*X.flatten() + 5 + np.random.randn(100)*10
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)
pred = model.predict(X_test)
print('MSE:', mean_squared_error(y_test, pred))
print('Coefficients:', model.coef_, 'Intercept:', model.intercept_)
```

Classification (Decision Tree example)

```
# Decision tree classifier example
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2, random_state=1)
clf = DecisionTreeClassifier(max_depth=3, random_state=1)
clf.fit(X_train, y_train)
pred = clf.predict(X_test)
print('Accuracy:', accuracy_score(y_test, pred))
print(export_text(clf, feature_names=iris.feature_names))
```

Algebraic Method for 2x2 Game (solve for mixed strategy)

```
# Solve 2x2 zero-sum game (Player A payoff matrix)
import numpy as np

A = np.array([[3,2],[1,4]], dtype=float)  # payoff to A

# Check for saddle point
min_row = A.min(axis=1).max()
max_col = A.max(axis=0).min()
if min_row == max_col:
    print('Saddle point value:', min_row)
else:
    # Solve for mixed strategy p for player A
    # Let p be prob of playing row1. Expected payoff when B chooses col1 and col2:
    # E1 = p*A11 + (1-p)*A21
    # E2 = p*A12 + (1-p)*A22
    # Solve E1 = E2
    import sympy as sp
    p = sp.symbols('p', real=True)
    A11,A12,A21,A22 = A[0,0],A[0,1],A[1,0],A[1,1]
    sol = sp.solve(sp.Eq(p*A11 + (1-p)*A21, p*A12 + (1-p)*A22), p)
    p_val = float(sol[0])
    # Value of the game
    val = p_val*A11 + (1-p_val)*A21
    print('Player A plays row1 with probability p =', p_val)
    print('Value of game =', val)
```

Simplex Algorithm (basic implementation)

```
# Simplex using scipy.optimize (if available) otherwise a simple tableau method
try:
    from scipy.optimize import linprog
    c = [-1, -2]  # maximize x1 + 2x2 -> minimize -c
```

```
    A = [[1,2],[1,1]]
    b = [20,12]
    res = linprog(c, A_ub=A, b_ub=b, bounds=(0,None), method='highs')
    if res.success:
        print('Optimal (x1,x2)=', res.x, 'Max Z =', -res.fun)
    else:
        print('Linprog failed:', res.message)
except Exception as e:
    print('scipy not available, fallback needed. Error:', e)
```

Linear Programming in PuLP (example)

```
# PuLP example
try:
    import pulp
    prob = pulp.LpProblem('MaxZ', pulp.LpMaximize)
    x1 = pulp.LpVariable('x1', lowBound=0)
    x2 = pulp.LpVariable('x2', lowBound=0)
    prob += x1 + 2*x2
    prob += x1 + 2*x2 <= 20
    prob += x1 + x2 <= 12
    prob.solve()
    print('Status:', pulp.LpStatus[prob.status])
    print('x1=', pulp.value(x1), 'x2=', pulp.value(x2))
except Exception as e:
    print('PuLP not installed:', e)
```

Two-person zero-sum game without saddle point (solve by linear programming)

```
# Solve a zero-sum game (no saddle point) via linear programming (player A maximizes)
import numpy as np
from scipy.optimize import linprog

A = np.array([[0,1,2],[3,1,0]], dtype=float)  # payoff to A, rows = strategies of A
m, n = A.shape
# Convert to LP: maximize v subject to A^T p >= v, sum p = 1, p >= 0
# Equivalent to minimize -v
# We'll transform to standard LP by shifting payoffs if necessary to be positive
min_val = A.min()
if min_val <= 0:
    A_shift = A - min_val + 1
else:
    A_shift = A.copy()
# Variables: p (m probs) and v (value)
# Use linprog to minimize -v (we'll put variables in order [p0..p_{m-1}, v])
c = [0]*m + [-1]
# Constraints: For each column j: sum_i A_shift[i,j]*p_i - v >= 0 -> -sum A_shift[i,j]*p_i + v <= 0
A_ub = []
b_ub = []
for j in range(n):
    row = [-A_shift[i,j] for i in range(m)] + [1]
    A_ub.append(row)
    b_ub.append(0)
# Probability sum constraint: sum p_i = 1
A_eq = [[1]*m + [0]]
b_eq = [1]
bounds = [(0,1)]*m + [(None,None)]
res = linprog(c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=bounds, method='highs')
if res.success:
    p = res.x[:m]
    v = res.x[-1] + min_val - 1  # shift back
    print('Mixed strategy for A:', p)
    print('Value of game:', v)
else:
    print('LP failed:', res.message)
```

Stock market simple analysis (moving average bull/bear)

```
# Simple moving-average based bull/bear indicator (using yfinance)
try:
    import yfinance as yf
    df = yf.download('RELIANCE.NS', period='6mo', interval='1d')
    df['MA20'] = df['Close'].rolling(20).mean()
    df['MA50'] = df['Close'].rolling(50).mean()
    latest = df.iloc[-1]
    signal = 'Bull' if latest['MA20'] > latest['MA50'] else 'Bear'
    print('Latest signal:', signal)
except Exception as e:
    print('yfinance not available or internet blocked:', e)
```

End of document. Copy the code blocks into .py files to run locally. Some snippets require additional packages (scipy, pulp, yfinance, sympy).