In [1]:
```
import pandas as pd

# --- STEP 1: Load SEC Submission + Numeric Data ---
sub = pd.read_csv("sub.txt", sep='\t', low_memory=False)
num = pd.read_csv("num.txt", sep='\t', low_memory=False)

# --- STEP 2: Filter for Financial Entities Including Investment Banks ---
relevant_sics = [6111, 6211, 6282, 6719, 6726, 6799]  # Investment banks, brokers,
institutions = sub[sub['sic'].isin(relevant_sics)]

# --- STEP 3: Filter Numeric Values ---
num_filtered = num[num['adsh'].isin(institutions['adsh'])]

tags_needed = [
    'Revenues', 'RevenueFromContractWithCustomerExcludingAssessedTax', 'SalesRevenu
    'AssetsCurrent', 'Assets',
    'LiabilitiesCurrent', 'Liabilities',
    'InterestExpense', 'InterestExpenseOperating', 'InterestAndDebtExpense',
    'LongTermDebt', 'DebtLongtermAndShorttermCombinedAmount',
    'StockholdersEquity', 'StockholdersEquityIncludingPortionAttributableToNoncontr
]
num_filtered = num_filtered[num_filtered['tag'].isin(tags_needed)]

# --- STEP 4: Pivot and Merge ---
pivot_df = num_filtered.pivot_table(index='adsh', columns='tag', values='value', ag
merged = institutions[['adsh', 'name', 'sic']]
final_df = pd.merge(merged, pivot_df, on='adsh')
final_df = final_df.rename(columns={'name': 'Counterparty_Name'})
```

In [3]:
```
# --- STEP 5: Fallback for Missing Financial Fields ---
def safe_combine(df, cols):
    valid_cols = [col for col in cols if col in df.columns]
    if not valid_cols:
        return pd.Series([None] * len(df), index=df.index)
    result = df[valid_cols[0]]
    for col in valid_cols[1:]:
        result = result.combine_first(df[col])
    return result

final_df['Revenue'] = safe_combine(final_df, ['Revenues', 'RevenueFromContractWithC
final_df['Debt'] = safe_combine(final_df, ['LongTermDebt', 'DebtLongtermAndShortter
final_df['Equity'] = safe_combine(final_df, ['StockholdersEquity', 'StockholdersEqu
final_df['Interest_Expense'] = safe_combine(final_df, ['InterestExpense', 'Interest
final_df['Current_Assets'] = safe_combine(final_df, ['AssetsCurrent', 'Assets'])
final_df['Current_Liabilities'] = safe_combine(final_df, ['LiabilitiesCurrent', 'Li
final_df['Total_Assets'] = safe_combine(final_df, ['Assets'])
final_df['Total_Liabilities'] = safe_combine(final_df, ['Liabilities'])
final_df['Retained_Earnings'] = safe_combine(final_df, ['RetainedEarningsAccumulate
final_df['Operating_Income'] = safe_combine(final_df, ['OperatingIncomeLoss'])


# --- STEP 6: Impute Missing Financial Values ---
financial_cols = ['Revenue', 'Debt', 'Equity', 'Interest_Expense', 'Current_Assets'
```

In [5]:
```
for col in financial_cols:
    final_df[col] = final_df.groupby('sic')[col].transform(lambda x: x.fillna(x.med
for col in financial_cols:
    final_df[col] = final_df[col].fillna(final_df[col].median())

final_df = final_df[final_df['Debt'] >= 0]
```

In [7]:
```
# --- STEP 7: Add Counterparty ID and Map Sector from SIC ---
final_df['Counterparty_ID'] = ['C' + str(i + 1).zfill(3) for i in range(len(final_d

sic_map = {
    6111: 'Credit Agency / Investment Bank',
    6211: 'Broker',
    6282: 'Asset Manager',
    6719: 'Holding Company',
    6726: 'Investment Office',
    6799: 'Investor / PE'
}
final_df['Sector'] = final_df['sic'].map(sic_map)

# --- STEP 8: Impute Sector Based on Name if SIC Wasn't Mapped ---
def infer_sector(name):
    name = str(name).upper()
    if 'BROKER' in name or 'SECURITIES' in name:
        return 'Broker'
    elif 'ASSET' in name or 'INVESTMENT' in name or 'FUND' in name:
        return 'Asset Manager'
    elif 'BANK' in name or 'CAPITAL' in name or 'MORTGAGE' in name or 'CREDIT' in n
        return 'Credit Agency / Investment Bank'
    elif 'HOLDING' in name or 'HOLDINGS' in name:
        return 'Holding Company'
    elif 'PARTNER' in name or 'PARTNERS' in name or 'EQUITY' in name or 'VENTURE' i
        return 'Investor / PE'
    return 'Other'

final_df['Sector'] = final_df.apply(
    lambda row: row['Sector'] if pd.notna(row['Sector']) else infer_sector(row['Cou
    axis=1
)

final_df['Z1'] = (final_df['Current_Assets'] - final_df['Current_Liabilities']) / f
final_df['Z2'] = final_df['Retained_Earnings'] / final_df['Total_Assets']
final_df['Z3'] = final_df['Operating_Income'] / final_df['Total_Assets']
final_df['Z4'] = (final_df['Total_Assets'] - final_df['Total_Liabilities']) / final
final_df['Z5'] = final_df['Revenue'] / final_df['Total_Assets']

# --- STEP 9: Final Output ---
result = final_df[['Counterparty_ID', 'Counterparty_Name', 'Revenue', 'Debt', 'Equi
                   'Interest_Expense', 'Current_Assets', 'Current_Liabilities', 'Se

# Preview
display(result.head())
```

	Counterparty_ID	Counterparty_Name	Revenue	Debt	Equity	Interest
0	C001	FRANKLIN RESOURCES INC	7.390000e+07	9.167300e+09	-4.503000e+08	2.31
1	C002	FEDERAL NATIONAL MORTGAGE ASSOCIATION FANNIE MAE	2.431590e+08	6.878500e+10	7.768200e+10	9.08
2	C003	SEI INVESTMENTS CO	0.000000e+00	9.970000e+08	2.131828e+09	4.46
3	C004	SCHWAB CHARLES CORP	4.187000e+09	9.970000e+08	2.000000e+07	7.17
4	C005	RAYMOND JAMES FINANCIAL INC	8.870000e+08	9.970000e+08	1.167300e+10	4.46

2. Financial Ratio Analysis

In [9]:
```
import numpy as np
# Safeguard denominators
final_df["Equity"] = final_df["Equity"].replace(0, np.nan)



final_df.loc[:, "Debt_to_Equity"] = final_df["Debt"] / final_df["Equity"]
final_df.loc[:, "Interest_Coverage"] = final_df["Revenue"] / final_df["Interest_Exp
final_df.loc[:, "Current_Ratio"] = final_df["Current_Assets"] / final_df["Current_L



# Define the ratio columns to check
ratio_cols = ["Debt_to_Equity", "Interest_Coverage", "Current_Ratio"]

# Keep only rows where all ratio columns are finite
final_df = final_df[np.isfinite(final_df[ratio_cols]).all(axis=1)]
```

3. Internal Rating Assignment

In [11]:
```
def assign_rating(row):
    # --- Distress Override: Negative Equity Scenario ---
    if row["Debt_to_Equity"] < 0:
        return "CCC"  # Firm is technically insolvent

    score = 0

    # --- Debt to Equity Scoring ---
    if row["Debt_to_Equity"] < 1.5:
        score += 2
    elif row["Debt_to_Equity"] < 2.5:
```

```
        score += 1

    # --- Interest Coverage Scoring ---
    if row["Interest_Coverage"] > 5:
        score += 2
    elif row["Interest_Coverage"] > 2:
        score += 1

    # --- Current Ratio Scoring ---
    if row["Current_Ratio"] > 1.5:
        score += 2
    elif row["Current_Ratio"] > 1.0:
        score += 1

    # --- Map to Rating Scale ---
    ratings = ["CCC", "B", "BB", "BBB", "A", "AA", "AAA"]
    return ratings[min(score, len(ratings) - 1)]

final_df["Internal_Rating"] = final_df.apply(assign_rating, axis=1)
```

In [13]:

```
import numpy as np

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

# Define possible categorical values
product_types = ['Loan', 'Bond', 'Repo', 'Derivative', 'Credit Card', 'Line of Cred
collateral_types = ['Gov Bonds', 'Corporate Bonds', 'Cash', 'Real Estate', 'None']
seniority_levels = ['Senior Secured', 'Senior Unsecured', 'Subordinated']

# Simulate additional LGD-relevant variables
n = len(final_df)
final_df['Exposure_Amount'] = np.random.uniform(1e6, 20e6, n).round(2)
final_df['Product_Type'] = np.random.choice(product_types, n)
final_df['Collateral_Type'] = np.random.choice(collateral_types, n)
final_df['Collateral_Value'] = np.random.uniform(0, 20e6, n).round(2)
final_df['Haircut_%'] = np.where(final_df['Collateral_Type'] == 'None', 1.0, np.ran
final_df['Seniority'] = np.random.choice(seniority_levels, n)
final_df['Recovery_Lag_Months'] = np.random.choice([3, 6, 12, 18], n)




# Calculate Net Collateral Value
final_df['Net_Collateral'] = final_df['Collateral_Value'] * (1 - final_df['Haircut_

# Compute Collateral Coverage Ratio (CCR)
final_df['CCR'] = final_df['Net_Collateral'] / final_df['Exposure_Amount']


# Function to assign base LGD from product type
def base_lgd_from_product(product):
    return {
        'Loan': 0.45,         # Partially collateralized
        'Bond': 0.60,         # Often unsecured or subordinated
        'Repo': 0.08,         # Fully collateralized, low LGD
        'Derivative': 0.15,   # Netting + collateral reduce LGD
```

```
        'Credit Card': 0.90,  # Unsecured retail
        'Line of Credit': 0.85  # Unsecured revolving
    }.get(product, 0.50)  # Fallback default


def adjust_lgd_from_collateral(collateral):
    return {
        'Cash': -0.05,
        'Gov Bonds': -0.04,
        'Corporate Bonds': -0.02,
        'Real Estate': 0.00,
        'None': 0.20
    }.get(collateral, 0.00)

# Function to adjust LGD based on seniority
def adjust_lgd_from_seniority(level):
    return {
        'Senior Secured': -0.10,
        'Senior Unsecured': 0.00,
        'Subordinated': 0.10
    }.get(level, 0.00)

# LGD adjustment based on CCR tier
def adjust_lgd_from_ccr(ccr):
    if ccr >= 1.0:
        return -0.15  # over-collateralized
    elif ccr >= 0.75:
        return -0.10
    elif ccr >= 0.5:
        return -0.05
    elif ccr >= 0.25:
        return 0.00
    else:
        return 0.10  # low or no collateral coverage

# Sector-based LGD adjustments
def adjust_lgd_from_sector(sector):
    if "bank" in sector.lower():
        return -0.05
    elif "hedge" in sector.lower():
        return 0.10
    elif "asset manager" in sector.lower():
        return 0.05
    elif "broker" in sector.lower():
        return 0.00
    else:
        return 0.00

# Calculate full institutional LGD using all adjustments
final_df['LGD_Institutional_Enhanced'] = final_df.apply(
    lambda row: min(
        max(
            base_lgd_from_product(row['Product_Type']) +
            adjust_lgd_from_collateral(row['Collateral_Type']) +
            adjust_lgd_from_seniority(row['Seniority']) +
            adjust_lgd_from_ccr(row['CCR']) +
```

```
            adjust_lgd_from_sector(row['Sector']),
            0.0
        ),
        1.0
    ),
    axis=1
).round(2)
```

In [ ]:
```
final_df
```

In [15]:
```
 #Moody's DRD-style PD mapping
rating_pd_map = {
    "AAA": 0.0001, "AA": 0.0002, "A": 0.0005, "BBB": 0.002,
    "BB": 0.01, "B": 0.05, "CCC": 0.20, "CC": 0.30, "C": 0.50, "D": 1.0
}
final_df['Mapped_PD'] = final_df['Internal_Rating'].map(rating_pd_map)
```

In [17]:
```
X_counterparty = final_df[['Z1', 'Z2', 'Z3', 'Z4', 'Z5']].replace([np.inf, -np.inf]
```

In [19]:
```
# Train logistic regression on American Bankruptcy data (latest year)

import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, roc_auc_score


bankruptcy_df = pd.read_csv("american_bankruptcy.csv")

# Recalculate Z-score variables
bankruptcy_df['Z1'] = (bankruptcy_df['X1'] - bankruptcy_df['X14']) / bankruptcy_df[
bankruptcy_df['Z2'] = bankruptcy_df['X15'] / bankruptcy_df['X10']
bankruptcy_df['Z3'] = bankruptcy_df['X12'] / bankruptcy_df['X10']
bankruptcy_df['Z4'] = (bankruptcy_df['X10'] - bankruptcy_df['X17']) / bankruptcy_df
bankruptcy_df['Z5'] = bankruptcy_df['X9'] / bankruptcy_df['X10']
bankruptcy_df['Altman_Z'] = (
    1.2 * bankruptcy_df['Z1'] +
    1.4 * bankruptcy_df['Z2'] +
    3.3 * bankruptcy_df['Z3'] +
    0.6 * bankruptcy_df['Z4'] +
    1.0 * bankruptcy_df['Z5']
)

# Ratios for rating
bankruptcy_df['Debt_to_Equity'] = bankruptcy_df['X11'] / bankruptcy_df['X15']
bankruptcy_df['Interest_Coverage'] = bankruptcy_df['X16'] / bankruptcy_df['X13']
bankruptcy_df['Current_Ratio'] = bankruptcy_df['X1'] / bankruptcy_df['X14']

# Clean extreme/inf values
ratio_cols = ['Debt_to_Equity', 'Interest_Coverage', 'Current_Ratio']
bankruptcy_df = bankruptcy_df[np.isfinite(bankruptcy_df[ratio_cols]).all(axis=1)]
bankruptcy_df = bankruptcy_df[bankruptcy_df['Debt_to_Equity'] >= 0]

# Assign internal rating
def assign_internal_rating(row):
```

```
    if row['Debt_to_Equity'] < 0:
        return "CCC"
    score = 0
    if row["Debt_to_Equity"] < 1.5:
        score += 2
    elif row["Debt_to_Equity"] < 2.5:
        score += 1
    if row["Interest_Coverage"] > 5:
        score += 2
    elif row["Interest_Coverage"] > 2:
        score += 1
    if row["Current_Ratio"] > 1.5:
        score += 2
    elif row["Current_Ratio"] > 1.0:
        score += 1
    ratings = ["CCC", "B", "BB", "BBB", "A", "AA", "AAA"]
    return ratings[min(score, len(ratings) - 1)]

bankruptcy_df['Internal_Rating'] = bankruptcy_df.apply(assign_internal_rating, axis

# Map Moody's-style PDs
rating_pd_map = {
    "AAA": 0.0001, "AA": 0.0002, "A": 0.0005, "BBB": 0.002,
    "BB": 0.01, "B": 0.05, "CCC": 0.20, "CC": 0.30, "C": 0.50, "D": 1.0
}
bankruptcy_df['Mapped_PD'] = bankruptcy_df['Internal_Rating'].map(rating_pd_map)

# Keep only latest record per company
latest_panel_df = bankruptcy_df.sort_values("year").drop_duplicates(subset="company


# Train logistic regression
features = ['Z1', 'Z2', 'Z3', 'Z4', 'Z5']
X_train = latest_panel_df[features].replace([np.inf, -np.inf], np.nan).fillna(0)
y_train = latest_panel_df['status_label'].apply(lambda x: 1 if x == 'failed' else 0
log_reg = LogisticRegression(max_iter=1000, class_weight='balanced')
log_reg.fit(X_train, y_train)

# Predict PD for counterparty dataset
final_df['PD_Logistic'] = log_reg.predict_proba(X_counterparty)[:, 1]

# Output preview
display(final_df[['Counterparty_ID', 'Counterparty_Name', 'Internal_Rating', 'Mappe
```

	Counterparty_ID	Counterparty_Name	Internal_Rating	Mapped_PD	PD_Logistic
9	C010	OPPENHEIMER HOLDINGS INC	BB	0.0100	0.637765
39	C039	TRILLER GROUP INC.	CCC	0.2000	0.588207
10	C011	GOLDMAN SACHS GROUP INC	CCC	0.2000	0.557811
4	C005	RAYMOND JAMES FINANCIAL INC	BB	0.0100	0.548658
81	C081	VIRTU FINANCIAL, INC.	BB	0.0100	0.540025
14	C015	MORGAN STANLEY	B	0.0500	0.538643
1	C002	FEDERAL NATIONAL MORTGAGE ASSOCIATION FANNIE MAE	BB	0.0100	0.538238
88	C088	APOLLO GLOBAL MANAGEMENT, INC.	A	0.0005	0.538097
22	C022	FEDERAL HOME LOAN MORTGAGE CORP	BB	0.0100	0.537319
52	C052	BLACKSTONE INC.	A	0.0005	0.536410

In [21]:

```
# --- STEP 1: Calculate Altman Z-Score Variables ---
final_df['Z1'] = (final_df['Current_Assets'] - final_df['Current_Liabilities']) / f
final_df['Z2'] = final_df['Retained_Earnings'] / final_df['Total_Assets']
final_df['Z3'] = final_df['Operating_Income'] / final_df['Total_Assets']
final_df['Z4'] = (final_df['Total_Assets'] - final_df['Total_Liabilities']) / final
final_df['Z5'] = final_df['Revenue'] / final_df['Total_Assets']

# --- STEP 2: Compute Altman Z-Score ---
final_df['Altman_Z'] = (
    1.2 * final_df['Z1'] +
    1.4 * final_df['Z2'] +
    3.3 * final_df['Z3'] +
    0.6 * final_df['Z4'] +
    1.0 * final_df['Z5']
)

# --- STEP 3: Assign Z-Zone Based on Altman Z-Score ---
def zscore_zone(z):
    if z < 1.8:
        return 'distress'
    elif z <= 3.0:
        return 'grey'
    else:
        return 'safe'

final_df['Z_Zone'] = final_df['Altman_Z'].apply(zscore_zone)

# --- STEP 4: Compute Final Weighted PD ---
```

```
def weighted_pd(row):
    if row['Z_Zone'] == 'safe':
        return 0.8 * row['Mapped_PD'] + 0.2 * row['PD_Logistic']
    elif row['Z_Zone'] == 'grey':
        return 0.5 * row['Mapped_PD'] + 0.5 * row['PD_Logistic']
    else:  # distress
        return 0.3 * row['Mapped_PD'] + 0.7 * row['PD_Logistic']

final_df['Final_PD'] = final_df.apply(weighted_pd, axis=1)

# Show preview
final_df[['Counterparty_ID', 'Counterparty_Name', 'Altman_Z', 'Z_Zone', 'Mapped_PD'
```

Out[21]:

	Counterparty_ID	Counterparty_Name	Altman_Z	Z_Zone	Mapped_PD	PD_Logistic
39	C039	TRILLER GROUP INC.	-4.369921	distress	0.20	0.588207
10	C011	GOLDMAN SACHS GROUP INC	-678.068327	distress	0.20	0.557811
9	C010	OPPENHEIMER HOLDINGS INC	-2579.017325	distress	0.01	0.637765
48	C048	FEDERAL HOME LOAN BANK OF SAN FRANCISCO	-0.017571	distress	0.20	0.526945
49	C049	FEDERAL HOME LOAN BANK OF TOPEKA	0.126492	distress	0.20	0.525647
71	C071	FEDERAL HOME LOAN BANK OF NEW YORK	0.141466	distress	0.20	0.525486
8	C009	FEDERAL AGRICULTURAL MORTGAGE CORP	0.246106	distress	0.20	0.524915
80	C080	ROBINHOOD MARKETS, INC.	0.985795	distress	0.20	0.509104
14	C015	MORGAN STANLEY	-20.435926	distress	0.05	0.538643
4	C005	RAYMOND JAMES FINANCIAL INC	-2514.254284	distress	0.01	0.548658

In [35]:

```
final_df.to_csv("df_sample.csv")
```

In [23]:

```
# ----------------------
# Step 1: Map Exposure Category
# ----------------------
exposure_type_map = {
    'Loan': 'Term',
    'Bond': 'Term',
    'Repo': 'Other',
```

```
    'Derivative': 'Other',
    'Credit Card': 'Revolving',
    'Line of Credit': 'Revolving'
}
final_df['Exposure_Category'] = final_df['Product_Type'].map(exposure_type_map).fil

# ----------------------
# Step 2: Assign credit conversion factor and Undrawn Limits for Revolving
# ----------------------
final_df['CCF'] = final_df['Exposure_Category'].map({
    'Revolving': 0.75,
    'Term': 1.0
}).fillna(1.0)

final_df['Undrawn_Limit'] = np.where(
    final_df['Exposure_Category'] == 'Revolving',
    0.25 * final_df['Assets'].fillna(0),
    0
)

# ----------------------
# Step 3: Term Loan Amortized EAD Calculation
# ----------------------
loan_term_months = 60
annual_rate = 0.06
monthly_rate = annual_rate / 12

def monthly_payment(principal, r, n):
    if principal == 0 or r == 0:
        return 0
    return (principal * r * (1 + r)**n) / ((1 + r)**n - 1)

def remaining_principal(p, r, n, ttd):
    if p == 0 or r == 0:
        return 0
    return p * ((1 + r)**n - (1 + r)**ttd) / ((1 + r)**n - 1)

final_df['Exposure_Amount'] = final_df['Exposure_Amount'].fillna(0)
final_df['Monthly_Installment'] = final_df['Exposure_Amount'].apply(
    lambda x: monthly_payment(x, monthly_rate, loan_term_months)
)

np.random.seed(42)
final_df['Time_to_Default'] = final_df['Final_PD'].apply(
    lambda pd: np.random.randint(1, min(loan_term_months, int((1 - pd) * loan_term_
)

final_df['EAD_Term_Amortized'] = final_df.apply(
    lambda row: remaining_principal(row['Exposure_Amount'], monthly_rate, loan_term
    if row['Exposure_Category'] == 'Term' else 0,
    axis=1
)

# ----------------------
# Step 4: Standardized approach for counterparty risk EAD Calculation for Derivativ
# ----------------------
```

```python
def calculate_saccr_ead(row):
    alpha = 1.4
    exposure = row['Exposure_Amount'] if not pd.isna(row['Exposure_Amount']) else 0
    collateral = row['Net_Collateral'] if not pd.isna(row['Net_Collateral']) else 0
    haircut = row['Haircut_%'] if not pd.isna(row['Haircut_%']) else 0.10
    rc = max(exposure - collateral, 0)
    pfe = exposure * haircut
    return round(alpha * (rc + pfe), 2)

final_df['EAD_SACCR'] = final_df.apply(
    lambda row: calculate_saccr_ead(row) if row['Exposure_Category'] == 'Other' els
    axis=1
)

# ----------------------
# Step 5: Final EAD Column
# ----------------------
final_df['EAD'] = np.where(
    final_df['Exposure_Category'] == 'Term',
    final_df['EAD_Term_Amortized'],
    np.where(
        final_df['Exposure_Category'] == 'Revolving',
        final_df['Exposure_Amount'] + final_df['CCF'] * (final_df['Undrawn_Limit']
        final_df['EAD_SACCR']
    )
)
```

In [32]:

```python
# Set regulatory floor values
pd_floor = 0.0005  # 0.05% minimum PD
lgd_floor = 0.10   # 10% minimum LGD

# Apply the floors
final_df['Final_PD_Floored'] = final_df['Final_PD'].apply(lambda x: max(x, pd_floor
final_df['LGD_Enhanced_Floored'] = final_df['LGD_Institutional_Enhanced'].apply(lam

# Recalculate Expected Loss using floored values
final_df['Expected_Loss_Floored'] = (
    final_df['Final_PD_Floored'] * final_df['LGD_Enhanced_Floored'] * final_df['EAD
).round(2)
```

In [37]:

```python
final_df.head()
```

Out[37]:

	adsh	Counterparty_Name	sic	Assets	AssetsCurrent	DebtLongtermAr
0	0000038777-25-000017	FRANKLIN RESOURCES INC	6282.0	3.246450e+10	NaN	
1	0000310522-25-000199	FEDERAL NATIONAL MORTGAGE ASSOCIATION FANNIE MAE	6111.0	2.040000e+11	NaN	
2	0000350894-25-000028	SEI INVESTMENTS CO	6211.0	2.520003e+09	169867000.0	
3	0000316709-25-000010	SCHWAB CHARLES CORP	6211.0	1.586000e+09	NaN	
4	0000720005-25-000025	RAYMOND JAMES FINANCIAL INC	6211.0	2.700000e+07	NaN	

5 rows × 66 columns

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: