For this GWP, I focused on the scenario of lending at a fixed rate for an unsecured purchase. In GWP 2, I had previously worked with macroeconomic data to analyze correlations and perform Granger Causality tests. This time, I expanded my analysis by incorporating three additional variables: the Unemployment Rate, the Charge-Off Rate on Credit Card Loans, and the Personal Saving Rate. I employed KDE plots and histograms to visualize the distributions of these variables and created heatmaps to examine the correlations between macroeconomic variables and credit data variables. Finally, I used Linear Regression and Random Forest models to predict credit data based on macroeconomic variables, aiming to assess their predictive power.

In [1]:
```
import requests
import quandl
import datetime
from datetime import date
import pandas as pd
import numpy as np
import seaborn as sns
import math as mt
from matplotlib import pyplot as plt
import yfinance as yf
import matplotlib.dates as mdates
yf.pdr_override()
%matplotlib inline
```

```
yfinance: pandas_datareader support is deprecated & semi-broken so will be removed i
n a future verison. Just use yfinance.
```

In [2]:
```
def macroeconomic_datas(file, columns_rename):
    df = pd.read_csv(file)
    df.set_index("DATE", inplace=True)
    df.columns = columns_rename
    print(df.shape)
    print(df.head())
    return df
```

In [3]:
```
delinquency_rate_df = macroeconomic_datas('Credit Card unsecured loan/Delinquency R
                    ['Delinquency Rate'])
```

```
(80, 1)
           Delinquency Rate
DATE
1/1/2004               4.21
4/1/2004               4.15
7/1/2004               4.07
10/1/2004              4.03
1/1/2005               3.70
```

In [4]:
```
gdp_rate_df = macroeconomic_datas('Credit Card unsecured loan/Real GDP.csv', ['GDP'
```

```
(80, 1)
            GDP
DATE
1/1/2004   2.3
4/1/2004   3.1
7/1/2004   3.8
10/1/2004  4.1
1/1/2005   4.5
```

In [5]:
```
effective_rate_df = macroeconomic_datas('Credit Card unsecured loan/Effective Rates
```

```
(80, 1)
            Effective Rate
DATE
1/1/2004          1.003333
4/1/2004          1.010000
7/1/2004          1.433333
10/1/2004         1.950000
1/1/2005          2.470000
```

In [6]:
```
interest_rate_df = macroeconomic_datas('Credit Card unsecured loan/Commercial Bank
                   ['interest rate'])
```

```
(80, 1)
            interest rate
DATE
1/1/2004            12.67
4/1/2004            12.69
7/1/2004            13.02
10/1/2004           12.51
1/1/2005            12.21
```

In [7]:
```
market_yield_df = macroeconomic_datas('Credit Card unsecured loan/Market Yield on U
                  ['Market Yield'])
```

```
(80, 1)
            Market Yield
DATE
1/1/2004        4.877903
4/1/2004        5.355645
7/1/2004        5.068281
10/1/2004       4.874839
1/1/2005        4.764098
```

In [8]:
```
personal_loan_rate_df = macroeconomic_datas('Credit Card unsecured loan/Finance Rat
                  ['Loan Interest Rate'])
```

```
(80, 1)
            Loan Interest Rate
DATE
1/1/2004                 11.88
4/1/2004                 11.80
7/1/2004                 12.02
10/1/2004                11.84
1/1/2005                 12.01
```

In [9]:
```
unemployment_rate = macroeconomic_datas('Credit Card unsecured loan/Unemployement R
```

```
(80, 1)
                UR
DATE
1/1/2004   5.700000
4/1/2004   5.600000
7/1/2004   5.433333
10/1/2004  5.433333
1/1/2005   5.300000
```

In [10]:
```
personal_saving_rate = macroeconomic_datas("Credit Card unsecured loan/Personal Sav
```

```
(80, 1)
            ps_rate
DATE
1/1/2004   4.566667
4/1/2004   5.033333
7/1/2004   4.500000
10/1/2004  4.500000
1/1/2005   2.500000
```

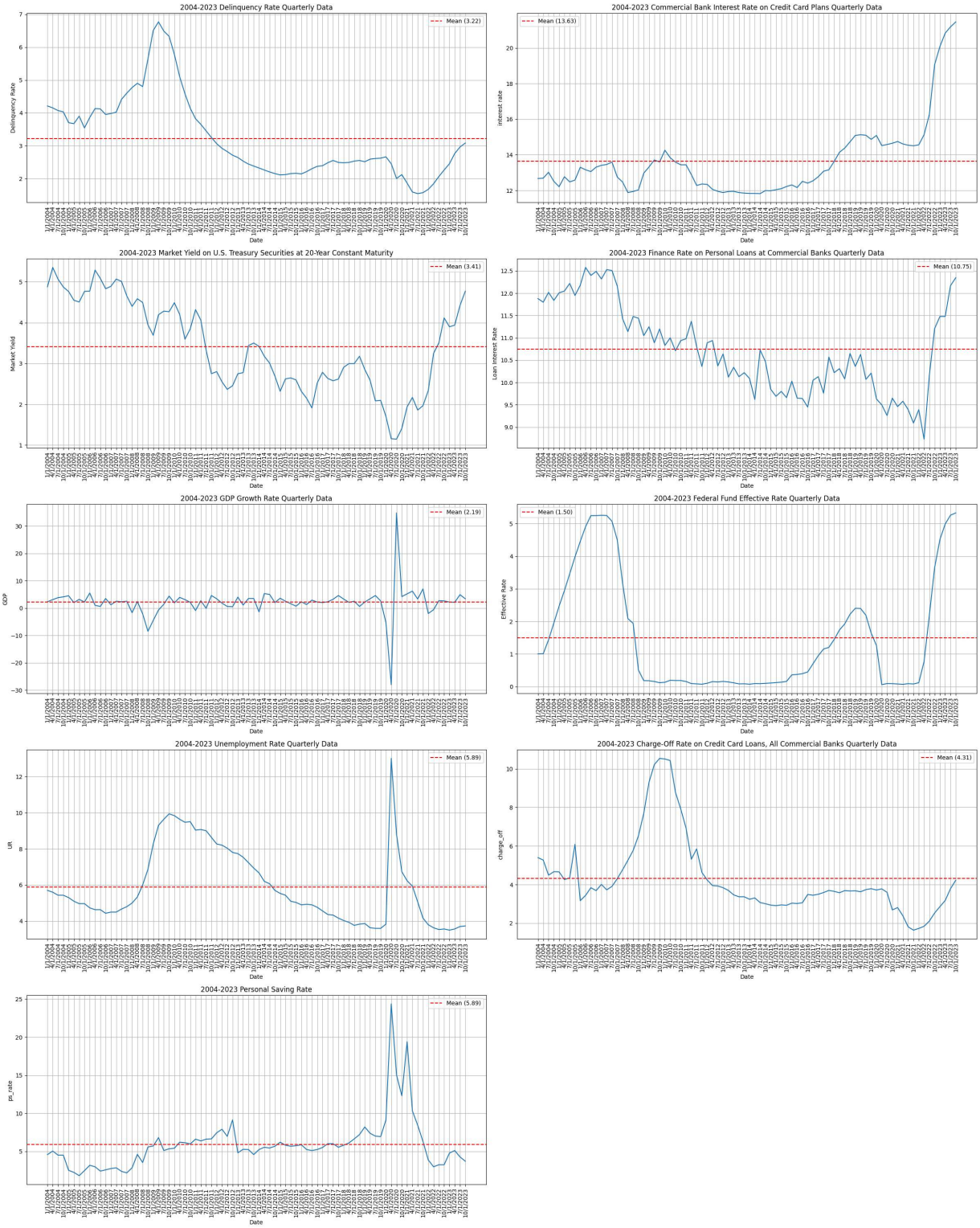
In [11]:
```
charge_off = macroeconomic_datas('Credit Card unsecured loan/Charge-Off Rate on Cre
                                 ['charge_off'])
```

```
(80, 1)
           charge_off
DATE
1/1/2004         5.40
4/1/2004         5.27
7/1/2004         4.49
10/1/2004        4.67
1/1/2005         4.66
```

In this script, we're creating a function to plot multiple economic datasets on separate graphs. Each plot shows a different economic indicator over time, with a red dashed line marking the average value for that indicator. After defining the function, we use it to visualize data like delinquency rates, interest rates, market yields, personal loan rates, GDP growth, Charge off Rate on Credit Card Loans, Unemployment rate, Personal saving Rate and federal fund rates from 2004 to 2023.

In [12]:
```
import math

def plot_economic_data(*dfs_titles):
    num_plots = len(dfs_titles)
    nrows = math.ceil(num_plots / 2)
    ncols = 2 if num_plots > 1 else 1

    fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(24, 6 * nrows))

    if num_plots == 1:
        axes = [axes]
    else:
        axes = axes.flatten()

    for i, (df, title) in enumerate(dfs_titles):
        mean_value = df.iloc[:, 0].mean()
```

```
        axes[i].plot(df.index, df.values)
        axes[i].set_title(title)
        axes[i].set_xlabel('Date')
        axes[i].set_ylabel(df.columns[0])
        axes[i].axhline(y=mean_value, color='red', linestyle='--', label=f'Mean ({m
        axes[i].legend()
        axes[i].grid(True)
        for tick in axes[i].get_xticklabels():
            tick.set_rotation(90)

    for j in range(i + 1, len(axes)):
        fig.delaxes(axes[j])

    plt.tight_layout()
    plt.show()

plot_economic_data(
    (delinquency_rate_df, '2004-2023 Delinquency Rate Quarterly Data'),
    (interest_rate_df, '2004-2023 Commercial Bank Interest Rate on Credit Card Plan
    (market_yield_df, '2004-2023 Market Yield on U.S. Treasury Securities at 20-Yea
    (personal_loan_rate_df, '2004-2023 Finance Rate on Personal Loans at Commercial
    (gdp_rate_df, '2004-2023 GDP Growth Rate Quarterly Data'),
    (effective_rate_df, '2004-2023 Federal Fund Effective Rate Quarterly Data'),
    (unemployment_rate, '2004-2023 Unemployment Rate Quarterly Data'),
    (charge_off, '2004-2023 Charge-Off Rate on Credit Card Loans, All Commercial Ba
    (personal_saving_rate, '2004-2023 Personal Saving Rate')
)
```



In [13]:
```
def volatility(*dfs):
    processed_dfs = []
    for df in dfs:
        processed_df = (df.pct_change().dropna()) * 100
        processed_dfs.append(processed_df)
    merged_df = pd.concat(processed_dfs, axis=1)
    return merged_df

volatility(delinquency_rate_df,
           interest_rate_df,
```

```
          market_yield_df,
          personal_loan_rate_df,
          gdp_rate_df,
          effective_rate_df,
          unemployment_rate,
          personal_saving_rate,
         charge_off)
```

Out[13]:

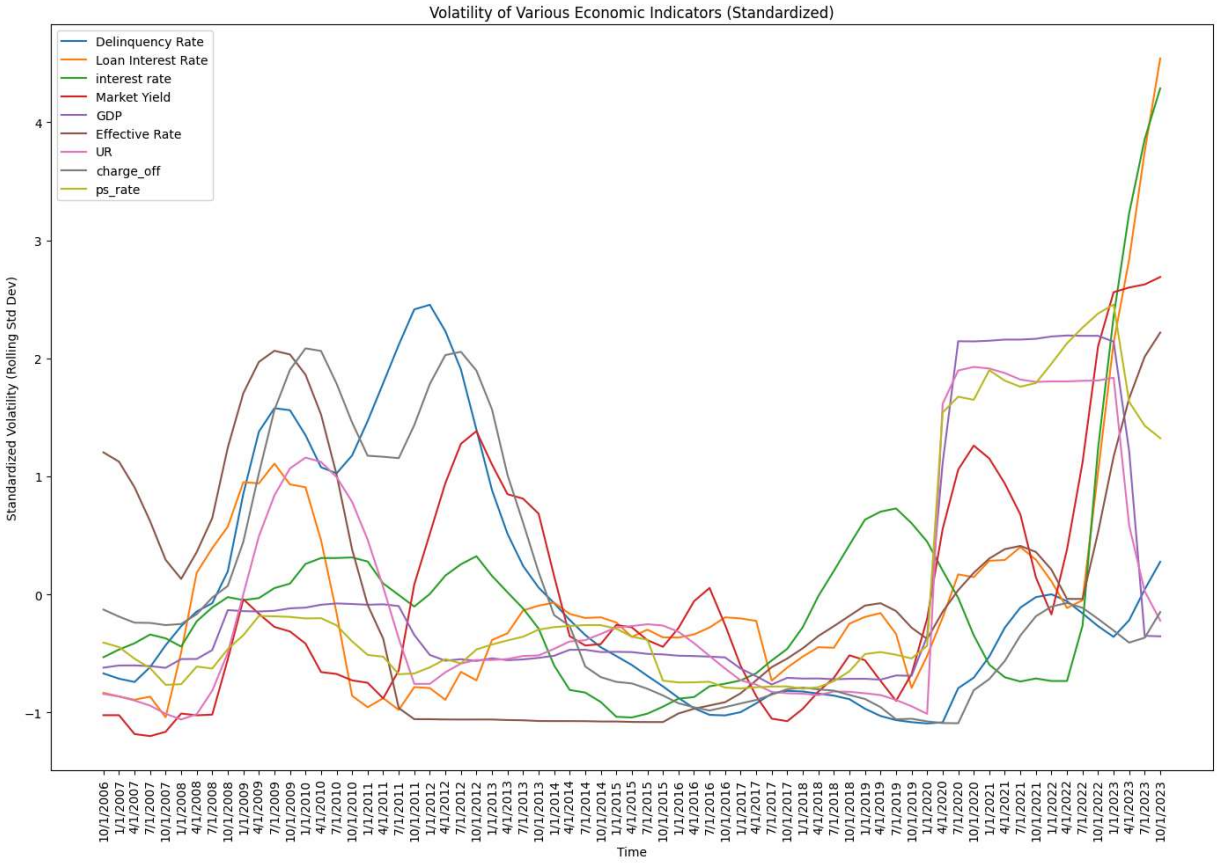
DATE	Delinquency Rate	interest rate	Market Yield	Loan Interest Rate	GDP	Effective Rate	UR
4/1/2004	-1.425178	0.157853	9.794002	-0.673401	34.782609	0.664452	-1.754386
7/1/2004	-1.927711	2.600473	-5.365626	1.864407	22.580645	41.914191	-2.976190
10/1/2004	-0.982801	-3.917051	-3.816729	-1.497504	7.894737	36.046512	0.000000
1/1/2005	-8.188586	-2.398082	-2.271672	1.435811	9.756098	26.666667	-2.453988
4/1/2005	-0.810811	4.586405	-4.520233	0.333056	-55.555556	19.163293	-3.773585
...	...	...	...	...	...	...	...
10/1/2022	9.708738	17.209588	17.514560	10.334646	-3.703704	66.818874	0.943396
1/1/2023	8.407080	5.348715	-5.275877	2.408564	-15.384615	23.631387	-1.869159
4/1/2023	12.653061	3.733201	0.997258	0.000000	-4.545455	10.479705	1.904762
7/1/2023	7.246377	1.679463	12.255179	6.010453	133.333333	5.410822	3.738318
10/1/2023	4.054054	1.321378	7.909219	1.479047	-30.612245	1.330798	0.900901

79 rows × 9 columns

In this script, we're creating a function to plot the volatility of several economic indicators. We calculate the rolling standard deviation (volatility) for each dataset and standardize the results to make them comparable. Then, we plot these standardized volatilities on a single graph to show how they change over time.

In [14]:
```
from sklearn.preprocessing import StandardScaler

def volatility_chart(*dfs, window=12):
    plt.figure(figsize=(14, 10))

    scaler = StandardScaler()
    for i, df in enumerate(dfs):
        volatility = df.rolling(window=window).std().dropna()
        standardized_volatility = scaler.fit_transform(volatility)

        plt.plot(volatility.index, standardized_volatility, label=df.columns[0])
```

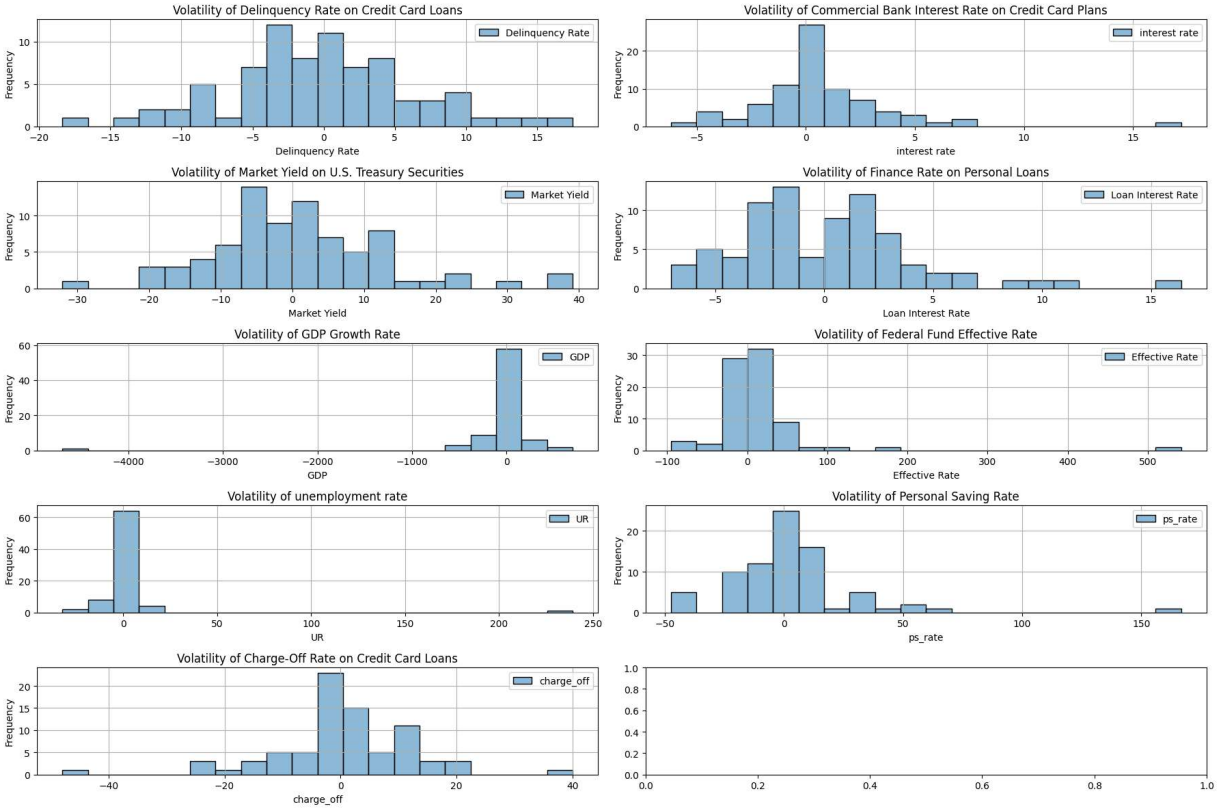
```
    plt.title('Volatility of Various Economic Indicators (Standardized)')
    plt.legend()
    plt.xlabel('Time')
    plt.ylabel('Standardized Volatility (Rolling Std Dev)')
    plt.xticks(rotation=90)
    plt.tight_layout()
    plt.show()

volatility_chart(
    delinquency_rate_df, personal_loan_rate_df, interest_rate_df,
    market_yield_df, gdp_rate_df, effective_rate_df, unemployment_rate,
    charge_off, personal_saving_rate
)
```



In this script, we are creating histograms to check the distribution of volatility for several economic indicators. Each histogram shows how frequently different levels of volatility occur for indicators like delinquency rates, interest rates, market yields, personal loan rates, GDP growth, Charge off Rate on Credit Card Loans, Unemployment rate, Personal saving Rate and the federal fund rate. This helps us understand the variability and distribution of these economic factors over time.

In [36]:
```
volatility_df = volatility(delinquency_rate_df,
                           interest_rate_df,
                           market_yield_df,
                           personal_loan_rate_df,
                           gdp_rate_df,
                           effective_rate_df,
                          unemployment_rate,
```

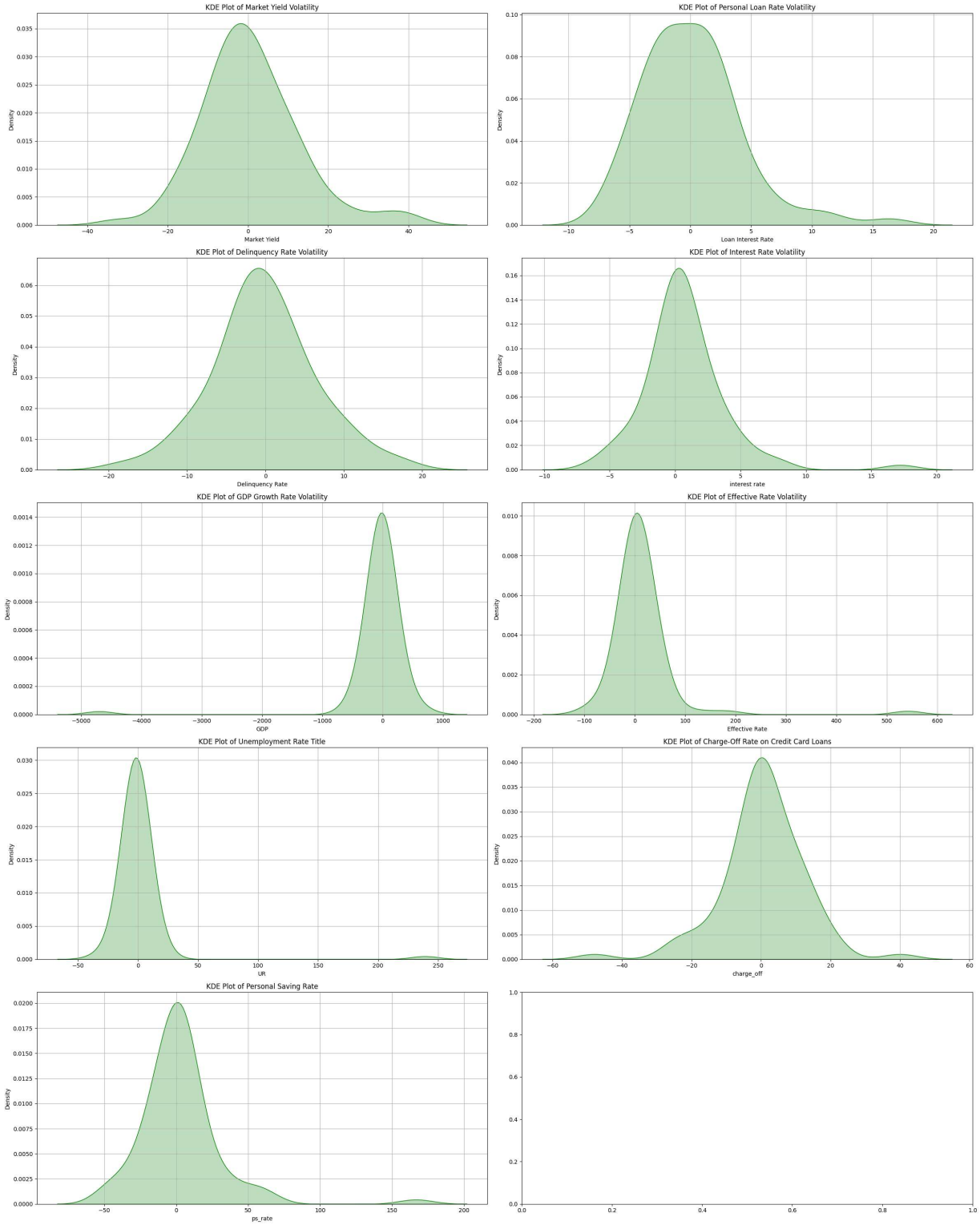
```
                          charge_off,
                          personal_saving_rate)

market_yield_title = 'Volatility of Market Yield on U.S. Treasury Securities'
personal_loan_rate_title = 'Volatility of Finance Rate on Personal Loans'
delinquency_rate_title = 'Volatility of Delinquency Rate on Credit Card Loans'
interest_rate_title = 'Volatility of Commercial Bank Interest Rate on Credit Card P
gdp_rate_title = 'Volatility of GDP Growth Rate'
effective_rate_title = 'Volatility of Federal Fund Effective Rate'
unemployment_rate_title = 'Volatility of unemployment rate'
charge_off_title = 'Volatility of Charge-Off Rate on Credit Card Loans'
personal_saving_rate_title = 'Volatility of Personal Saving Rate'

def plot_economic_distributions(*dfs_and_titles):
    num_plots = len(dfs_and_titles)
    ncols = 2
    nrows = (num_plots + 1) // ncols
    fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(18, 12))

    for idx, (df, title) in enumerate(dfs_and_titles):
        row = idx // ncols
        col = idx % ncols
        sns.histplot(df, bins=20, alpha=0.5, color='blue', edgecolor='black', ax=ax
        axes[row, col].set_title(title)
        axes[row, col].set_xlabel(df.columns[0])
        axes[row, col].set_ylabel('Frequency')
        axes[row, col].grid(True)

    plt.tight_layout()
    plt.show()

plot_data = [
    (volatility_df[['Delinquency Rate']], delinquency_rate_title),
    (volatility_df[['interest rate']], interest_rate_title),
    (volatility_df[['Market Yield']], market_yield_title),
    (volatility_df[['Loan Interest Rate']], personal_loan_rate_title),
    (volatility_df[['GDP']], gdp_rate_title),
    (volatility_df[['Effective Rate']], effective_rate_title),
    (volatility_df[['UR']], unemployment_rate_title),
    (volatility_df[['ps_rate']], personal_saving_rate_title),
    (volatility_df[['charge_off']], charge_off_title)
]

plot_economic_distributions(*plot_data)
```



In this script, we are creating KDE (Kernel Density Estimate) plots to visualize the distribution of volatility for several economic indicators. Each plot shows the density of volatility values, helping us understand the variability and concentration of different levels of volatility for indicators like market yield, personal loan rates, delinquency rates, interest rates, GDP growth, Unemployment rate, Personal saving Rate and the federal fund rate.

In [16]:
```
def plot_kde(*args):
    num_plots = len(args) // 3
    num_rows = (num_plots + 1) // 2

    fig, axes = plt.subplots(nrows=num_rows, ncols=2, figsize=(24, 6 * num_rows))

    for i in range(num_plots):
        df = args[i * 3]
        title = args[i * 3 + 1]
        col = args[i * 3 + 2]

        row = i // 2
        col_pos = i % 2

        sns.kdeplot(df[col], color='green', fill=True, ax=axes[row, col_pos])
        axes[row, col_pos].set_title(title)
        axes[row, col_pos].set_xlabel(col)
        axes[row, col_pos].set_ylabel('Density')
        axes[row, col_pos].grid(True)

    plt.tight_layout()
    plt.show()
```

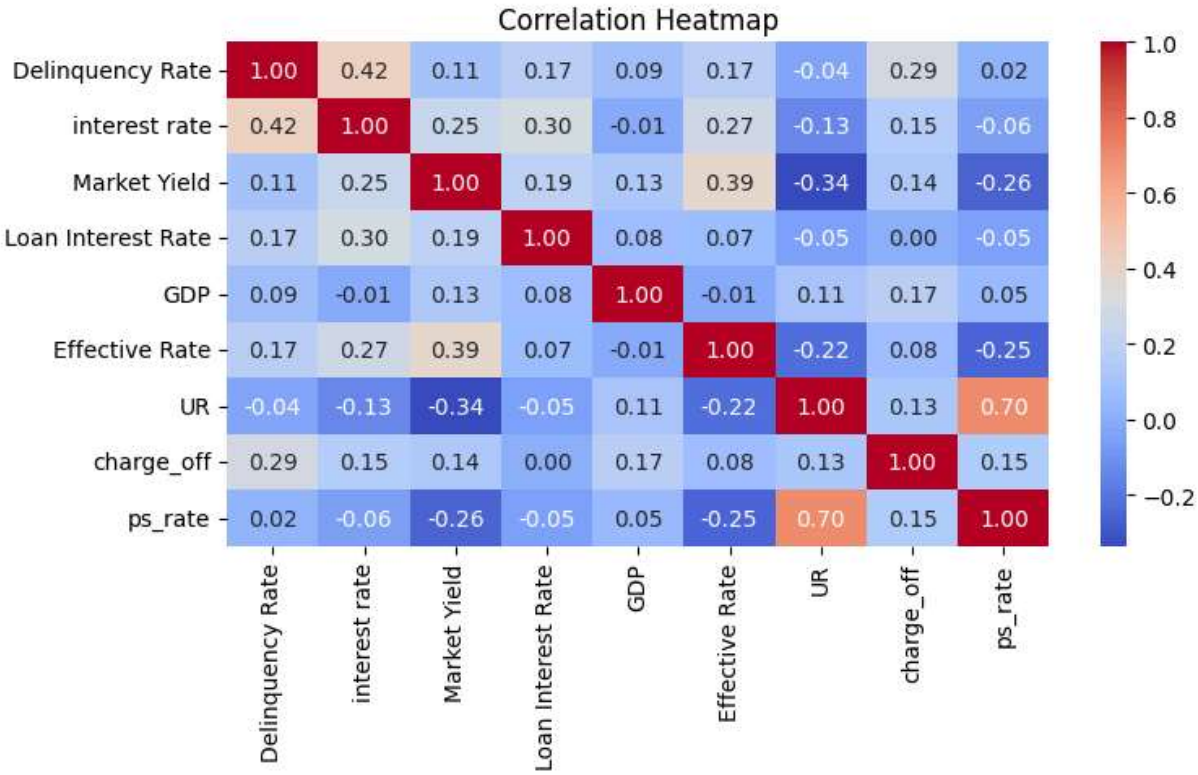
```
market_yield_title = 'KDE Plot of Market Yield Volatility'
personal_loan_rate_title = 'KDE Plot of Personal Loan Rate Volatility'
delinquency_rate_title = 'KDE Plot of Delinquency Rate Volatility'
interest_rate_title = 'KDE Plot of Interest Rate Volatility'
gdp_rate_title = 'KDE Plot of GDP Growth Rate Volatility'
effective_rate_title = 'KDE Plot of Effective Rate Volatility'
unemployment_rate_title = 'KDE Plot of Unemployment Rate Title'
charge_off_title = 'KDE Plot of Charge-Off Rate on Credit Card Loans'
personal_saving_rate_title = 'KDE Plot of Personal Saving Rate'

plot_kde(
    volatility_df, market_yield_title, 'Market Yield',
    volatility_df, personal_loan_rate_title, 'Loan Interest Rate',
    volatility_df, delinquency_rate_title, 'Delinquency Rate',
    volatility_df, interest_rate_title, 'interest rate',
    volatility_df, gdp_rate_title, 'GDP',
    volatility_df, effective_rate_title, 'Effective Rate',
    volatility_df, unemployment_rate_title, 'UR',
    volatility_df, charge_off_title, 'charge_off',
    volatility_df, personal_saving_rate_title, 'ps_rate'
)
```



In this script, we are creating a correlation heatmap to show the relationships between the volatilities of various economic indicators. The heatmap displays the correlation coefficients between each pair of indicators, helping us understand how closely related their volatilities are. This visualization makes it easy to see which indicators move together and which ones do not.

In [17]:
```
def corr(df):
    correlation_matrix = df.corr()
```

```
    plt.figure(figsize=(8, 5))
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
    plt.title('Correlation Heatmap')
    plt.xticks(rotation=90)
    plt.yticks(rotation=0)
    plt.tight_layout()
    plt.show()
corr(volatility_df)
```



In [18]:
```
volatility_df.describe().T
```

Out[18]:

	count	mean	std	min	25%	50%	75%
Delinquency Rate	79.0	-0.181663	6.556284	-18.367347	-3.729011	-0.401606	3.48334
interest rate	79.0	0.717489	3.170589	-6.181015	-0.779619	0.480110	1.76572
Market Yield	79.0	0.692021	12.300713	-32.119514	-6.453294	-0.340066	6.57734
Loan Interest Rate	79.0	0.129726	4.100288	-7.028754	-2.586773	-0.103627	2.03610
GDP	79.0	-69.359496	557.383551	-4700.000000	-68.333333	-26.086957	35.03836
Effective Rate	79.0	14.250041	70.219369	-95.238095	-10.565476	5.410822	19.60468
UR	79.0	1.218915	27.962336	-32.307692	-4.110770	-2.032520	0.52463
charge_off	79.0	0.469209	11.987077	-47.947455	-3.751700	0.000000	7.43786
ps_rate	79.0	2.849776	28.228654	-47.445255	-10.631347	1.176471	9.03983

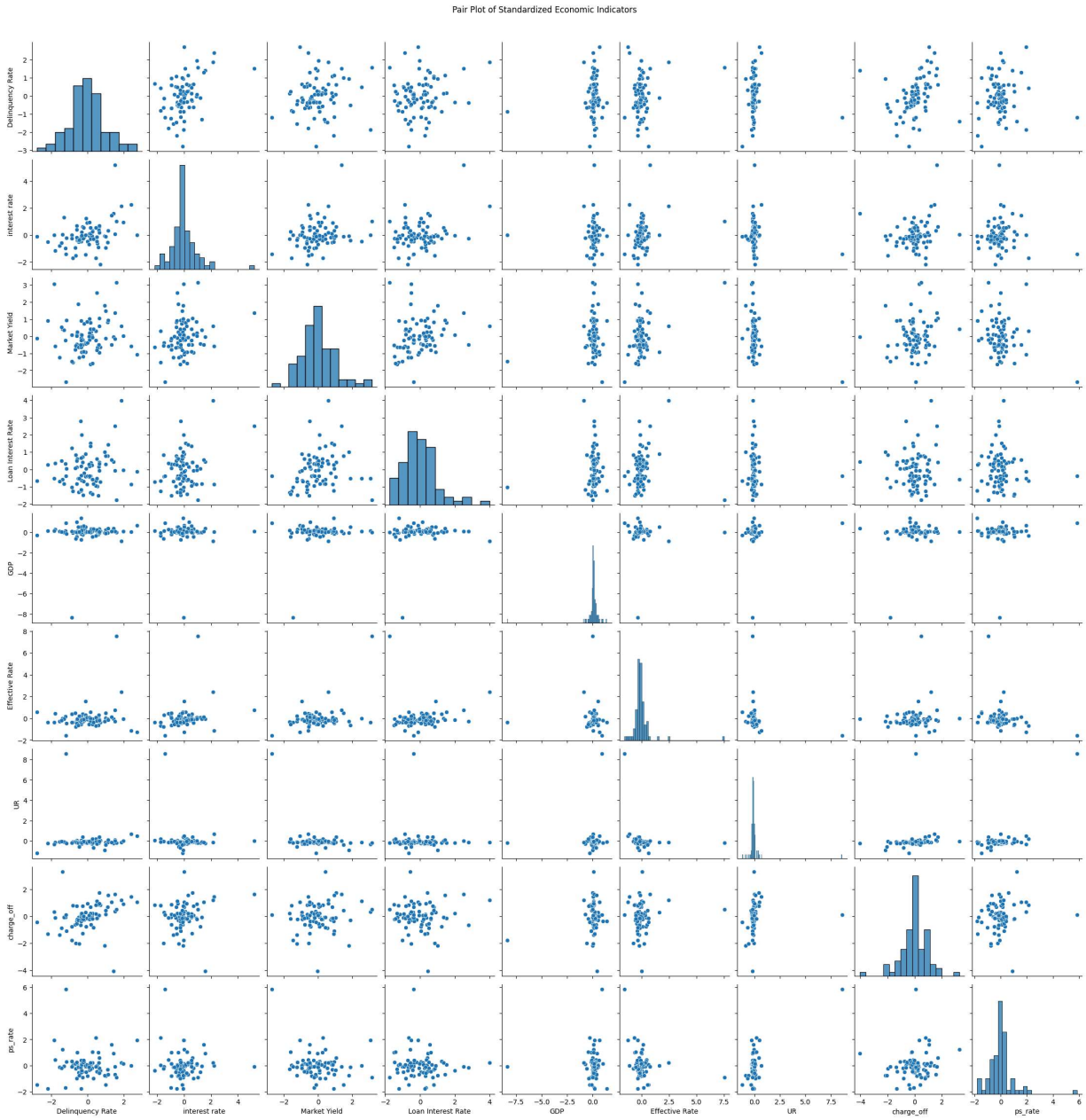
In [19]:

```
from scipy.stats import pearsonr

macro_data = ['Market Yield', 'GDP', 'Effective Rate', 'UR']
credit_data = ['Loan Interest Rate', 'Delinquency Rate', 'interest rate', 'charge_o

def calculate_correlations(volatility_df, data1, data2):
    correlations = {}
    p_values = {}

    for data1_col in data1:
        correlations[data1_col] = {}
        p_values[data1_col] = {}
        for data2_col in data2:
            corr, p_value = pearsonr(volatility_df[data1_col], volatility_df[data2_
            correlations[data1_col][data2_col] = corr
            p_values[data1_col][data2_col] = p_value

            print(f"{data1_col} vs {data2_col}: Correlation = {corr:.2f}, P-value =

            if p_value < 0.05:
                print("   The correlation is statistically significant.")
            else:
                print("   The correlation is not statistically significant.")

    return correlations, p_values

correlations, p_values = calculate_correlations(volatility_df, macro_data, credit_d
```

```
Market Yield vs Loan Interest Rate: Correlation = 0.19, P-value = 0.0905
   The correlation is not statistically significant.
Market Yield vs Delinquency Rate: Correlation = 0.11, P-value = 0.3493
   The correlation is not statistically significant.
Market Yield vs interest rate: Correlation = 0.25, P-value = 0.0265
   The correlation is statistically significant.
Market Yield vs charge_off: Correlation = 0.14, P-value = 0.2070
   The correlation is not statistically significant.
Market Yield vs ps_rate: Correlation = -0.26, P-value = 0.0203
   The correlation is statistically significant.
GDP vs Loan Interest Rate: Correlation = 0.08, P-value = 0.5030
   The correlation is not statistically significant.
GDP vs Delinquency Rate: Correlation = 0.09, P-value = 0.4475
   The correlation is not statistically significant.
GDP vs interest rate: Correlation = -0.01, P-value = 0.9279
   The correlation is not statistically significant.
GDP vs charge_off: Correlation = 0.17, P-value = 0.1395
   The correlation is not statistically significant.
GDP vs ps_rate: Correlation = 0.05, P-value = 0.6514
   The correlation is not statistically significant.
Effective Rate vs Loan Interest Rate: Correlation = 0.07, P-value = 0.5391
   The correlation is not statistically significant.
Effective Rate vs Delinquency Rate: Correlation = 0.17, P-value = 0.1273
   The correlation is not statistically significant.
Effective Rate vs interest rate: Correlation = 0.27, P-value = 0.0153
   The correlation is statistically significant.
Effective Rate vs charge_off: Correlation = 0.08, P-value = 0.5069
   The correlation is not statistically significant.
Effective Rate vs ps_rate: Correlation = -0.25, P-value = 0.0256
   The correlation is statistically significant.
UR vs Loan Interest Rate: Correlation = -0.05, P-value = 0.6878
   The correlation is not statistically significant.
UR vs Delinquency Rate: Correlation = -0.04, P-value = 0.7589
   The correlation is not statistically significant.
UR vs interest rate: Correlation = -0.13, P-value = 0.2413
   The correlation is not statistically significant.
UR vs charge_off: Correlation = 0.13, P-value = 0.2723
   The correlation is not statistically significant.
UR vs ps_rate: Correlation = 0.70, P-value = 0.0000
   The correlation is statistically significant.
```

In this script, we use pair plot a valuable tool for preliminary data analysis, allowing us to quickly assess potential correlations and the overall structure of the data. This visualization aids in understanding how different macroeconomic indicators and credit metrics interact with each other.

In [20]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
volatility_df_scaled = pd.DataFrame(scaler.fit_transform(volatility_df), columns=vo

sns.pairplot(volatility_df_scaled)
plt.suptitle('Pair Plot of Standardized Economic Indicators', y=1.02)
plt.show()
```



In [21]:
```
from statsmodels.tsa.stattools import adfuller
import statsmodels.api as sm

def adf_test(series, column_name):
    result = adfuller(series)
    print(f'ADF statistics for {column_name}: {result[0]}')
    print(f'p-value for {column_name}: {result[1]}')
    for key, value in result[4].items():
        print(f'Critical Value {key}: {value}')
    if result[1] <= 0.05:
        print(f"{column_name} is stationary")
        print('')
        return True
    else:
        print(f"{column_name} is not stationary")
        print('')
        return False
    print('')
```

```
def make_stationary(df):
    stationary_df = pd.DataFrame()
    for column in df.columns:
        series = df[column]
        is_stationary = adf_test(series, column)
        differencing_count = 0
        while not is_stationary:
            series = series.diff().dropna()
            differencing_count += 1
            is_stationary = adf_test(series, column)
            if differencing_count > 2:
                break
        stationary_df[column] = series
    return stationary_df

stationary_volatility_df = make_stationary(volatility_df)
```

```
ADF statistics for Delinquency Rate: -4.779812005023137
p-value for Delinquency Rate: 5.943415936862519e-05
Critical Value 1%: -3.517113604831504
Critical Value 5%: -2.8993754262546574
Critical Value 10%: -2.5869547797501644
Delinquency Rate is stationary

ADF statistics for interest rate: -4.394580117861085
p-value for interest rate: 0.0003034575211701177
Critical Value 1%: -3.5219803175527606
Critical Value 5%: -2.9014701097664504
Critical Value 10%: -2.58807215485756
interest rate is stationary

ADF statistics for Market Yield: -6.025436335607989
p-value for Market Yield: 1.4597643884033348e-07
Critical Value 1%: -3.518281134660583
Critical Value 5%: -2.899878185191432
Critical Value 10%: -2.5872229937594873
Market Yield is stationary

ADF statistics for Loan Interest Rate: -10.672882362816258
p-value for Loan Interest Rate: 4.1238444317242905e-19
Critical Value 1%: -3.517113604831504
Critical Value 5%: -2.8993754262546574
Critical Value 10%: -2.5869547797501644
Loan Interest Rate is stationary

ADF statistics for GDP: -8.721325247934276
p-value for GDP: 3.3793180285563605e-14
Critical Value 1%: -3.517113604831504
Critical Value 5%: -2.8993754262546574
Critical Value 10%: -2.5869547797501644
GDP is stationary

ADF statistics for Effective Rate: -2.7643709949537523
p-value for Effective Rate: 0.06354610172210438
Critical Value 1%: -3.526004646825607
Critical Value 5%: -2.9032002348069774
Critical Value 10%: -2.5889948363419957
Effective Rate is not stationary

ADF statistics for Effective Rate: -3.7726163814841334
p-value for Effective Rate: 0.003200245060072766
Critical Value 1%: -3.5274258688046647
Critical Value 5%: -2.903810816326531
Critical Value 10%: -2.5893204081632653
Effective Rate is stationary

ADF statistics for UR: -9.249553722096099
p-value for UR: 1.5051146981000688e-15
Critical Value 1%: -3.517113604831504
Critical Value 5%: -2.8993754262546574
Critical Value 10%: -2.5869547797501644
UR is stationary
```

```
ADF statistics for charge_off: -2.930414117688788
p-value for charge_off: 0.041923843403991974
Critical Value 1%: -3.5232835753964475
Critical Value 5%: -2.902030597326081
Critical Value 10%: -2.5883710883843123
charge_off is stationary

ADF statistics for ps_rate: -9.345206089592796
p-value for ps_rate: 8.583272418551063e-16
Critical Value 1%: -3.517113604831504
Critical Value 5%: -2.8993754262546574
Critical Value 10%: -2.5869547797501644
ps_rate is stationary
```

In [22]:

```python
from statsmodels.tsa.stattools import grangercausalitytests
import warnings
warnings.filterwarnings('ignore', category=FutureWarning)


macro_data = ['Market Yield', 'GDP', 'Effective Rate', 'UR']
credit_data = ['Loan Interest Rate', 'Delinquency Rate', 'interest rate', 'charge_o

def granger_causality_report(data, macro_vars, credit_vars, max_lag):
    report = {}
    best_results = []

    for macro_var in macro_vars:
        for credit_var in credit_vars:
            p_values = []
            for lag in range(1, max_lag + 1):
                result = grangercausalitytests(data[[credit_var, macro_var]], lag,
                p_value = result[lag][0]['ssr_ftest'][1]
                p_values.append(p_value)

            min_p_value = min(p_values)
            best_lag = p_values.index(min_p_value) + 1
            report[f"{macro_var}->{credit_var}"] = p_values

            if min_p_value < 0.05:
                best_results.append((macro_var, credit_var, min_p_value, best_lag))

    return report, best_results

max_lag = 20
report, best_results = granger_causality_report(volatility_df, macro_data, credit_d

df_report = pd.DataFrame(report, index=[f"Lag {lag}" for lag in range(1, max_lag +
df_report = df_report.transpose()

df_report.head(10)
```

Out[22]:

	Lag 1	Lag 2	Lag 3	Lag 4	Lag 5	Lag 6	Lag 7	Lag
Market Yield->Loan Interest Rate	0.000741	0.003794	0.006325	0.005550	0.014928	0.006217	0.003333	0.0012
Market Yield->Delinquency Rate	0.589909	0.707106	0.804014	0.318461	0.377809	0.413947	0.544547	0.5807
Market Yield->interest rate	0.132641	0.267493	0.304814	0.441035	0.524392	0.403560	0.032980	0.0677
Market Yield->charge_off	0.962081	0.943676	0.877189	0.948593	0.546240	0.197592	0.252478	0.4087
Market Yield->ps_rate	0.221649	0.453774	0.167601	0.229830	0.214634	0.215666	0.279312	0.2230
GDP->Loan Interest Rate	0.112115	0.232257	0.184422	0.383673	0.483017	0.624516	0.723320	0.8158
GDP->Delinquency Rate	0.806275	0.847796	0.930261	0.959220	0.978111	0.955098	0.872499	0.8683
GDP->interest rate	0.889139	0.893954	0.964739	0.983061	0.952087	0.982306	0.982407	0.9856
GDP->charge_off	0.597318	0.745993	0.979998	0.993114	0.992924	0.968426	0.965478	0.8181
GDP->ps_rate	0.360055	0.608368	0.757995	0.799272	0.253324	0.376444	0.434455	0.5508

In [23]:
```
best_results_df = pd.DataFrame(best_results, columns=['Macro Variable', 'Credit Var
best_results_df
```

Out[23]:

	Macro Variable	Credit Variable	Min P-Value	Best Lag
0	Market Yield	Loan Interest Rate	7.409808e-04	1
1	Market Yield	interest rate	3.297976e-02	7
2	Effective Rate	Loan Interest Rate	5.687922e-08	2
3	Effective Rate	Delinquency Rate	2.055089e-02	1
4	Effective Rate	interest rate	8.650323e-07	2
5	UR	Loan Interest Rate	6.626400e-04	11
6	UR	Delinquency Rate	8.500271e-06	3
7	UR	interest rate	8.946289e-06	18
8	UR	charge_off	5.160085e-04	10

In [31]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

macro_data = ['Market Yield', 'GDP', 'Effective Rate', 'UR']
credit_data = ['Loan Interest Rate', 'Delinquency Rate', 'interest rate', 'charge_o

X = volatility_df[macro_data]
results = {}

plt.figure(figsize=(18, 24))

for idx, target in enumerate(credit_data, start=1):
    y = volatility_df[[target]]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random
    model = LinearRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    results[target] = {
        'model': model,
        'mse': mse,
        'r2': r2
    }

    print(f'Prediction results for {target}:')
    print(f'Mean Squared Error: {mse}')
    print(f'R2 Score: {r2}\n')

loan_interest_model = results['Loan Interest Rate']['model']
```

```
Prediction results for Loan Interest Rate:
Mean Squared Error: 11.504840258198502
R2 Score: -0.3661824673362968

Prediction results for Delinquency Rate:
Mean Squared Error: 73.58824819555429
R2 Score: -0.5503007849651007

Prediction results for interest rate:
Mean Squared Error: 5.509468868711612
R2 Score: -0.6905663898317771

Prediction results for charge_off:
Mean Squared Error: 818.7480848574334
R2 Score: -8.780420979663052

Prediction results for ps_rate:
Mean Squared Error: 845.9393804285189
R2 Score: -0.09641715214942903

<Figure size 1800x2400 with 0 Axes>
```

In [39]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

macro_data = volatility_df[['Market Yield', 'GDP', 'Effective Rate', 'UR']]
credit_data = {
    'Loan Interest Rate': volatility_df['Loan Interest Rate'],
    'Delinquency Rate': volatility_df['Delinquency Rate'],
    'interest rate': volatility_df['interest rate'],
    'charge_off': volatility_df['charge_off'],
    'ps_rate': volatility_df['ps_rate']
}

n_estimators_range = range(10, 101, 10)
results = {target: {'n_estimators': [], 'MSE': [], 'R2': []} for target in credit_d

for n_estimators in n_estimators_range:
    for target, y in credit_data.items():
        X = macro_data
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra

        model = RandomForestRegressor(n_estimators=n_estimators, random_state=42)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        mse = mean_squared_error(y_test, y_pred)
        r2 = r2_score(y_test, y_pred)

        results[target]['n_estimators'].append(n_estimators)
        results[target]['MSE'].append(mse)
        results[target]['R2'].append(r2)

best_estimators = {}

for target in credit_data.keys():
    min_mse_index = results[target]['MSE'].index(min(results[target]['MSE']))
    best_estimators[target] = {
        'Best n_estimators': results[target]['n_estimators'][min_mse_index],
        'Best MSE': results[target]['MSE'][min_mse_index],
        'Best R2': results[target]['R2'][min_mse_index]
    }
for target, metrics in best_estimators.items():
    print(f"Best number of estimators for {target}:")
    print(f" - Number of Estimators: {metrics['Best n_estimators']}")
    print(f" - Best MSE: {metrics['Best MSE']}")
    print(f" - Best R2: {metrics['Best R2']}")
    print()
```

```
Best number of estimators for Loan Interest Rate:
 - Number of Estimators: 10
 - Best MSE: 15.2587844803052
 - Best R2: -0.8119576945017393

Best number of estimators for Delinquency Rate:
 - Number of Estimators: 70
 - Best MSE: 42.87420829703869
 - Best R2: 0.09675905586142397

Best number of estimators for interest rate:
 - Number of Estimators: 100
 - Best MSE: 2.590172007836602
 - Best R2: 0.20521236353666072

Best number of estimators for charge_off:
 - Number of Estimators: 20
 - Best MSE: 106.97491231933816
 - Best R2: -0.2778774034358238

Best number of estimators for ps_rate:
 - Number of Estimators: 30
 - Best MSE: 849.1000643525955
 - Best R2: -0.10051369635467111
```