```
! gdown --id 1BvQRvWXdbpXpOaSahEMovATmKQuQilQa
```

```
/usr/local/lib/python3.10/dist-packages/gdown/__main__.py:132: FutureWarning: Option `--id` was deprecated in version 4.3.1 and
  warnings.warn(
Downloading...
From: https://drive.google.com/uc?id=1BvQRvWXdbpXpOaSahEMovATmKQuQilQa
To: /content/SMA_data.csv
100% 101k/101k [00:00<00:00, 10.9MB/s]
```

```
pip install ruptures
```

```
Collecting ruptures
  Downloading ruptures-1.1.9-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (7.2 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from ruptures) (1.26.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from ruptures) (1.13.1)
Downloading ruptures-1.1.9-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 15.8 MB/s eta 0:00:00
Installing collected packages: ruptures
Successfully installed ruptures-1.1.9
```

Maneuver Detection Tools

```
import pandas as pd
import numpy as np
from scipy import signal
from scipy.fft import fft
from statsmodels.tsa.ar_model import AutoReg
from sklearn.preprocessing import PolynomialFeatures

df = pd.read_csv('SMA_data.csv')
df.info()

df['Datetime'] = pd.to_datetime(df['Datetime'])
df.set_index('Datetime', inplace=True)

# Rolling Statistics (Window-Based Features)
df['rolling_mean'] = df['SMA'].rolling(window=3).mean()
df['rolling_std'] = df['SMA'].rolling(window=3).std()
df['rolling_min'] = df['SMA'].rolling(window=3).min()
df['rolling_max'] = df['SMA'].rolling(window=3).max()

# Linear Trend Feature
X = np.arange(len(df)).reshape(-1, 1)
poly = PolynomialFeatures(degree=1)
X_poly = poly.fit_transform(X)
df['linear_trend'] = X_poly[:, 1]

# Exponential Moving Average
df['ema'] = df['SMA'].ewm(span=5, adjust=False).mean()

# Frequency Domain Features (Fourier Transform)
fft_values = fft(df['SMA'].values)
df['fft_real'] = np.real(fft_values)
df['fft_imag'] = np.imag(fft_values)

# Lag-Based Features
df['lag_1'] = df['SMA'].shift(1)
df['lag_2'] = df['SMA'].shift(2)

# Autoregressive Features
model = AutoReg(df['SMA'].dropna(), lags=2)
ar_model = model.fit()
df['ar_coef_1'] = ar_model.params[1]  # First AR coefficient
df['ar_coef_2'] = ar_model.params[2]  # Second AR coefficient

# Cumulative Features
df['cumulative_sum'] = df['SMA'].cumsum()
df['cumulative_max'] = df['SMA'].cummax()
df['cumulative_min'] = df['SMA'].cummin()

# Differencing and Change Features
df['diff_1'] = df['SMA'].diff(1)
df['diff_2'] = df['SMA'].diff(2)
df['pct_change'] = df['SMA'].pct_change()
df['direction'] = np.sign(df['SMA'].diff())

# Anomaly/Change Detection Features
```

```
# Anomaly/Change Detection Features
df['z_score'] = (df['SMA'] - df['SMA'].mean()) / df['SMA'].std()
df['outlier'] = np.abs(df['z_score']) > 2  # Mark points as outliers if Z-score > 2

# Time-Based Features
df['hour'] = df.index.hour
df['day_of_week'] = df.index.dayofweek
df['month'] = df.index.month

# Segment-Level Features (Segment by day)
df['date'] = df.index.date
daily_groups = df.groupby('date')['SMA']
df['daily_mean'] = daily_groups.transform('mean')
df['daily_std'] = daily_groups.transform('std')

# Drop temporary columns used for segmentation
df.drop(columns=['date'], inplace=True)

# ---- New Features ----

# SMA Velocity (First Derivative)
df['sma_velocity'] = df['SMA'].diff(1)  # First derivative of SMA

# Rolling Window SMA Differences (Difference between SMA and rolling mean)
df['rolling_diff'] = df['SMA'] - df['rolling_mean']

# Acceleration Features (Second Derivative)
df['sma_acceleration'] = df['sma_velocity'].diff(1)  # Second derivative of SMA

# Display the resulting DataFrame with extracted features
print(df.head())
```

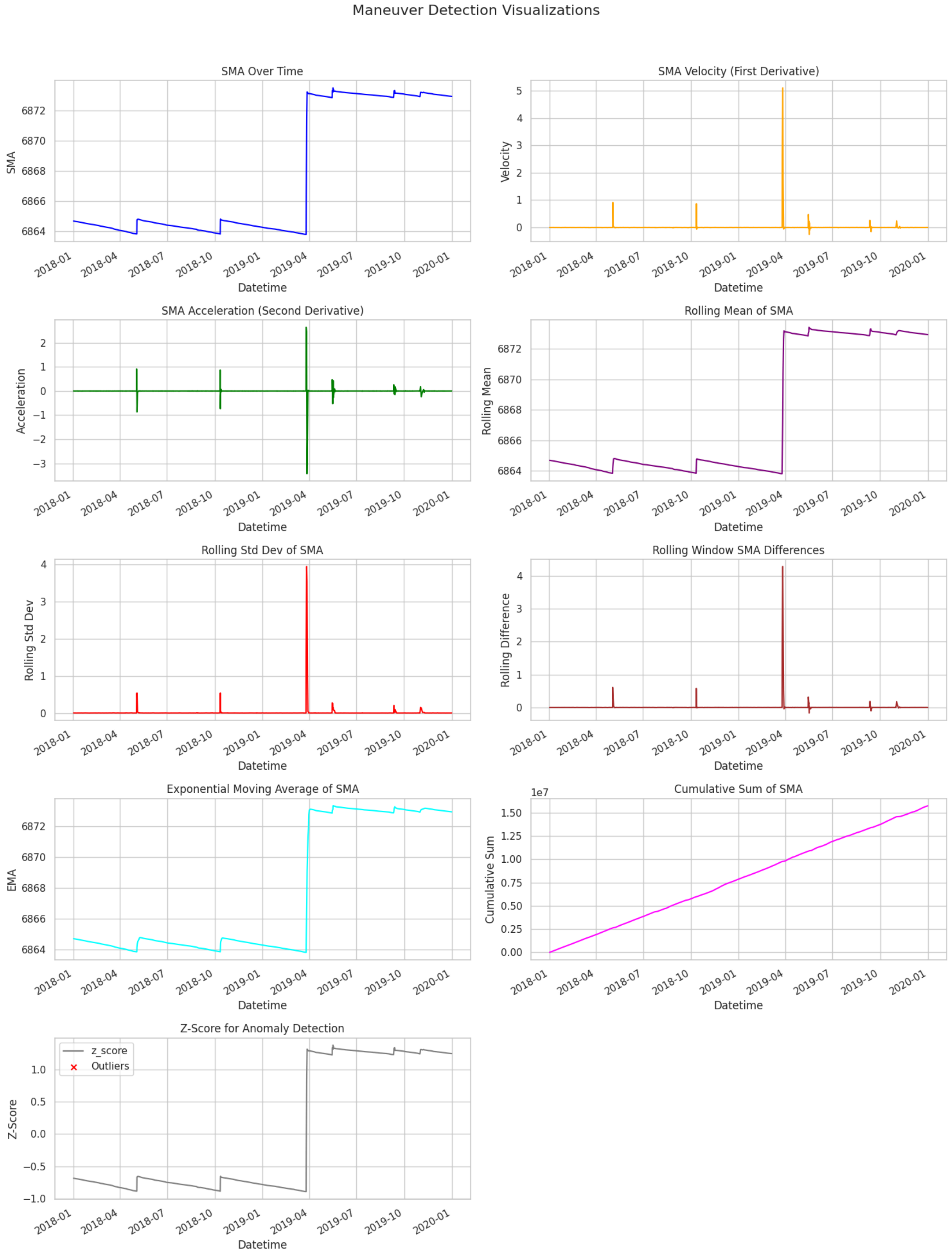
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2291 entries, 0 to 2290
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Datetime  2291 non-null   object
 1   SMA       2291 non-null   float64
dtypes: float64(1), object(1)
memory usage: 35.9+ KB
                                    SMA  rolling_mean  rolling_std  \
Datetime
2018-01-01 04:34:10.320672  6864.691463           NaN          NaN
2018-01-01 12:37:36.596064  6864.689664           NaN          NaN
2018-01-01 20:31:55.898112  6864.688585   6864.689904     0.001454
2018-01-02 05:42:49.014720  6864.684927   6864.687725     0.002483
2018-01-02 12:13:01.263360  6864.682858   6864.685457     0.002900

                            rolling_min  rolling_max  linear_trend  \
Datetime
2018-01-01 04:34:10.320672          NaN          NaN           0.0
2018-01-01 12:37:36.596064          NaN          NaN           1.0
2018-01-01 20:31:55.898112  6864.688585  6864.691463           2.0
2018-01-02 05:42:49.014720  6864.684927  6864.689664           3.0
2018-01-02 12:13:01.263360  6864.682858  6864.688585           4.0

                                    ema      fft_real     fft_imag  \
Datetime
2018-01-01 04:34:10.320672  6864.691463  1.573370e+07    -0.000000
2018-01-01 12:37:36.596064  6864.690863  2.240799e+03  5462.646902
2018-01-01 20:31:55.898112  6864.690104 -1.606697e+03  1550.686849
2018-01-02 05:42:49.014720  6864.688378  8.029930e+02   300.361710
2018-01-02 12:13:01.263360  6864.686538  3.296002e+01  1723.774794

                                  lag_1  ...   z_score  outlier  hour  \
Datetime                                 ...
2018-01-01 04:34:10.320672          NaN  ... -0.686760    False     4
2018-01-01 12:37:36.596064  6864.691463  ... -0.687183    False    12
2018-01-01 20:31:55.898112  6864.689664  ... -0.687437    False    20
2018-01-02 05:42:49.014720  6864.688585  ... -0.688297    False     5
2018-01-02 12:13:01.263360  6864.684927  ... -0.688783    False    12

                            day_of_week  month   daily_mean  daily_std  \
Datetime
2018-01-01 04:34:10.320672            0      1  6864.689904   0.001454
2018-01-01 12:37:36.596064            0      1  6864.689904   0.001454
2018-01-01 20:31:55.898112            0      1  6864.689904   0.001454
2018-01-02 05:42:49.014720            1      1  6864.683018   0.001834
2018-01-02 12:13:01.263360            1      1  6864.683018   0.001834

                            sma_velocity  rolling_diff  sma_acceleration
Datetime
2018-01-01 04:34:10.320672           NaN           NaN               NaN
2018-01-01 12:37:36.596064     -0.001799           NaN               NaN
2018-01-01 20:31:55.898112     -0.001079     -0.001319          0.000720
2018-01-02 05:42:49.014720     -0.003658     -0.002798         -0.002578
```

```
2018-01-02 12:13:01.263360     -0.002069     -0.002598          0.001589
```

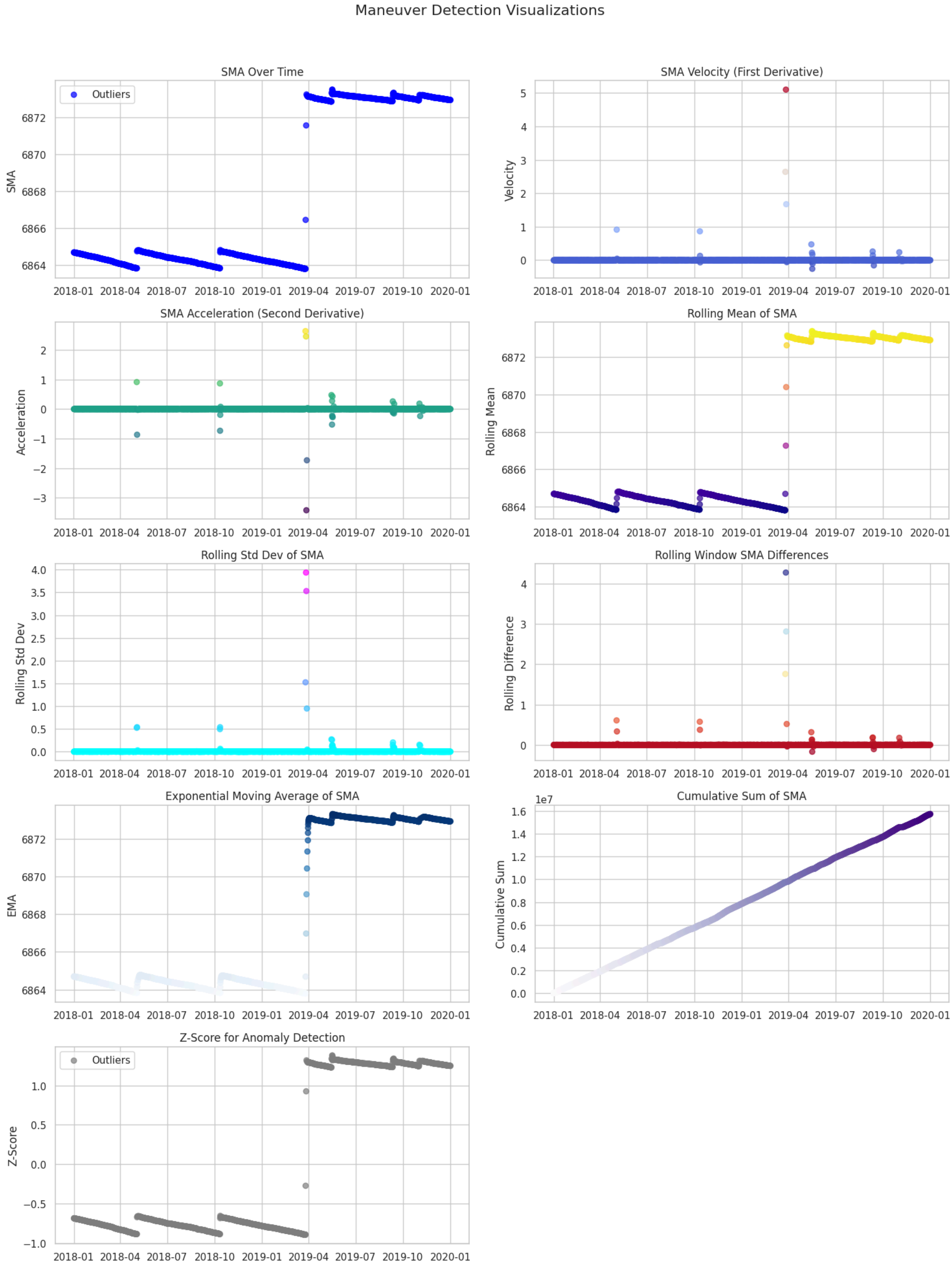
Visualizations

Line chart

```
import matplotlib.pyplot as plt
import seaborn as sns

# Set the plotting style
sns.set(style="whitegrid")

# Plot SMA and other features
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(15, 20))
fig.suptitle('Maneuver Detection Visualizations', fontsize=16)

# Plot the SMA values over time
df['SMA'].plot(ax=axes[0, 0], title='SMA Over Time', color='blue')
axes[0, 0].set_ylabel('SMA')

# Plot the SMA velocity (first derivative)
df['sma_velocity'].plot(ax=axes[0, 1], title='SMA Velocity (First Derivative)', color='orange')
axes[0, 1].set_ylabel('Velocity')

# Plot the SMA acceleration (second derivative)
df['sma_acceleration'].plot(ax=axes[1, 0], title='SMA Acceleration (Second Derivative)', color='green')
axes[1, 0].set_ylabel('Acceleration')

# Plot the rolling mean of SMA
df['rolling_mean'].plot(ax=axes[1, 1], title='Rolling Mean of SMA', color='purple')
axes[1, 1].set_ylabel('Rolling Mean')

# Plot the rolling standard deviation of SMA
df['rolling_std'].plot(ax=axes[2, 0], title='Rolling Std Dev of SMA', color='red')
axes[2, 0].set_ylabel('Rolling Std Dev')

# Plot the rolling window SMA differences
df['rolling_diff'].plot(ax=axes[2, 1], title='Rolling Window SMA Differences', color='brown')
axes[2, 1].set_ylabel('Rolling Difference')

# Plot the exponential moving average (EMA) of SMA
df['ema'].plot(ax=axes[3, 0], title='Exponential Moving Average of SMA', color='cyan')
axes[3, 0].set_ylabel('EMA')

# Plot the cumulative sum of SMA
df['cumulative_sum'].plot(ax=axes[3, 1], title='Cumulative Sum of SMA', color='magenta')
axes[3, 1].set_ylabel('Cumulative Sum')

# Plot the z-score (anomaly detection)
df['z_score'].plot(ax=axes[4, 0], title='Z-Score for Anomaly Detection', color='grey')
axes[4, 0].set_ylabel('Z-Score')

# Highlight the outliers based on the z-score
outliers = df[df['outlier'] == True]
axes[4, 0].scatter(outliers.index, outliers['z_score'], color='red', label='Outliers', marker='x')
axes[4, 0].legend()

# Hide the last empty subplot
axes[4, 1].axis('off')

# Adjust layout
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```


Maneuver Detection Visualizations
SMA Over Time
SMA
Datetime
SMA Velocity (First Derivative)
Velocity
Datetime
SMA Acceleration (Second Derivative)
Acceleration
Datetime
Rolling Mean of SMA
Rolling Mean
Datetime
Rolling Std Dev of SMA
Rolling Std Dev
Datetime
Rolling Window SMA Differences
Rolling Difference
Datetime
Exponential Moving Average of SMA
EMA
Datetime
Cumulative Sum of SMA
1e7
Cumulative Sum
Datetime
Z-Score for Anomaly Detection
z_score
Outliers
Z-Score
Datetime

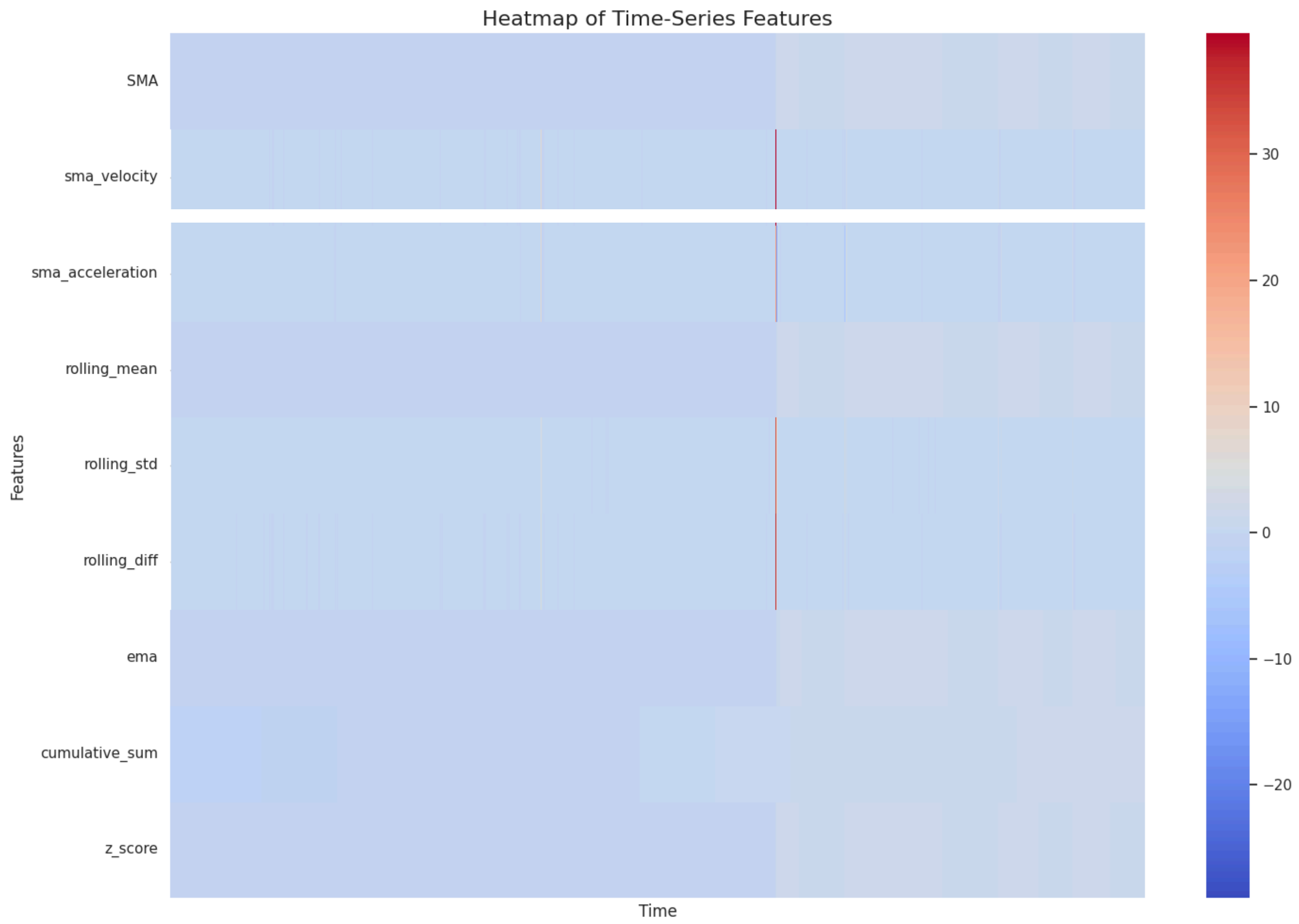
```
import matplotlib.pyplot as plt
import seaborn as sns

# Set the plotting style
sns.set(style="whitegrid")

# Plot SMA and other features
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(15, 20))
fig.suptitle('Maneuver Detection Visualizations', fontsize=16)

# Scatter plot for SMA values over time with color coding based on 'outlier' feature
axes[0, 0].scatter(df.index, df['SMA'], c=df['outlier'].map({True: 'red', False: 'blue'}), label='SMA', alpha=0.7)
axes[0, 0].set_title('SMA Over Time')
axes[0, 0].set_ylabel('SMA')
axes[0, 0].legend(['Outliers', 'SMA'])

# Scatter plot for SMA velocity (first derivative) with color coding
axes[0, 1].scatter(df.index, df['sma_velocity'], c=df['sma_velocity'], cmap='coolwarm', label='Velocity', alpha=0.7)
axes[0, 1].set_title('SMA Velocity (First Derivative)')
axes[0, 1].set_ylabel('Velocity')

# Scatter plot for SMA acceleration (second derivative) with color coding
axes[1, 0].scatter(df.index, df['sma_acceleration'], c=df['sma_acceleration'], cmap='viridis', label='Acceleration', alpha=0.7)
axes[1, 0].set_title('SMA Acceleration (Second Derivative)')
axes[1, 0].set_ylabel('Acceleration')

# Scatter plot for rolling mean of SMA with color coding
axes[1, 1].scatter(df.index, df['rolling_mean'], c=df['rolling_mean'], cmap='plasma', label='Rolling Mean', alpha=0.7)
axes[1, 1].set_title('Rolling Mean of SMA')
axes[1, 1].set_ylabel('Rolling Mean')

# Scatter plot for rolling standard deviation of SMA with color coding
axes[2, 0].scatter(df.index, df['rolling_std'], c=df['rolling_std'], cmap='cool', label='Rolling Std Dev', alpha=0.7)
axes[2, 0].set_title('Rolling Std Dev of SMA')
axes[2, 0].set_ylabel('Rolling Std Dev')

# Scatter plot for rolling window SMA differences with color coding
axes[2, 1].scatter(df.index, df['rolling_diff'], c=df['rolling_diff'], cmap='RdYlBu', label='Rolling Difference', alpha=0.7)
axes[2, 1].set_title('Rolling Window SMA Differences')
axes[2, 1].set_ylabel('Rolling Difference')

# Scatter plot for exponential moving average (EMA) of SMA with color coding
axes[3, 0].scatter(df.index, df['ema'], c=df['ema'], cmap='Blues', label='EMA', alpha=0.7)
axes[3, 0].set_title('Exponential Moving Average of SMA')
axes[3, 0].set_ylabel('EMA')

# Scatter plot for cumulative sum of SMA with color coding
axes[3, 1].scatter(df.index, df['cumulative_sum'], c=df['cumulative_sum'], cmap='Purples', label='Cumulative Sum', alpha=0.7)
axes[3, 1].set_title('Cumulative Sum of SMA')
axes[3, 1].set_ylabel('Cumulative Sum')

# Scatter plot for z-score with color coding for anomalies
axes[4, 0].scatter(df.index, df['z_score'], c=df['outlier'].map({True: 'red', False: 'grey'}), alpha=0.7, label='Z-Score')
axes[4, 0].set_title('Z-Score for Anomaly Detection')
axes[4, 0].set_ylabel('Z-Score')
axes[4, 0].legend(['Outliers', 'Z-Score'])

# Hide the last empty subplot
axes[4, 1].axis('off')

# Adjust layout
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```



Heat Map

```
import matplotlib.pyplot as plt
import seaborn as sns

# Set the plotting style
sns.set(style="whitegrid")

# Prepare the data for heatmap
# Select relevant columns for the heatmap
heatmap_data = df[['SMA', 'sma_velocity', 'sma_acceleration', 'rolling_mean', 'rolling_std', 'rolling_diff', 'ema', 'cumulative_su
```

```
# Normalize data for better heatmap visualization (optional)
normalized_data = (heatmap_data - heatmap_data.mean()) / heatmap_data.std()

# Create a figure with subplots
fig, ax = plt.subplots(figsize=(15, 10))

# Plot the heatmap
sns.heatmap(normalized_data.T, cmap='coolwarm', ax=ax, cbar=True, xticklabels=False)

# Customize the plot
ax.set_title('Heatmap of Time-Series Features', fontsize=16)
ax.set_ylabel('Features')
ax.set_xlabel('Time')

# Adjust layout
plt.tight_layout()
plt.show()
```



Data Preprocessing

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy.stats import zscore
```

```
# Load the dataset
# Replace 'dataset.csv' with your actual dataset file path
df = pd.read_csv('SMA_data.csv', parse_dates=['Datetime'])

# 1. Handling Missing Data
# Interpolating missing values
df['SMA'] = df['SMA'].interpolate(method='linear')

# 2. Time-Based Feature Engineering
df['Year'] = df['Datetime'].dt.year
df['Month'] = df['Datetime'].dt.month
df['Day'] = df['Datetime'].dt.day
df['DayOfWeek'] = df['Datetime'].dt.dayofweek
df['Hour'] = df['Datetime'].dt.hour

# Create time differences in seconds
df['Time_Diff'] = df['Datetime'].diff().dt.total_seconds()

# 3. SMA-Based Feature Engineering
df['SMA_Diff'] = df['SMA'].diff()  # Difference between consecutive SMA values
df['SMA_Pct_Change'] = df['SMA'].pct_change()  # Percentage change
df['SMA_CumSum'] = df['SMA_Diff'].cumsum()  # Cumulative sum of SMA differences

# 4. Data Normalization/Scaling
# Standardization (Z-score normalization)
scaler_standard = StandardScaler()
df['SMA_Standardized'] = scaler_standard.fit_transform(df[['SMA']])

# Min-Max Scaling
scaler_minmax = MinMaxScaler()
df['SMA_MinMax'] = scaler_minmax.fit_transform(df[['SMA']])

# 5. Handling Time Gaps
# Resampling to a regular interval (e.g., hourly)
df_resampled = df.set_index('Datetime').resample('1H').mean().interpolate()  # Resample to hourly and interpolate missing values

# Alternatively, flag irregular time gaps without resampling
df['Irregular_Time_Gap'] = df['Time_Diff'] > (df['Time_Diff'].mean() + 3 * df['Time_Diff'].std())

# 6. Outlier Detection and Removal
# Z-score based outlier detection
df['SMA_Zscore'] = zscore(df['SMA'])
df['Outlier_Flag'] = np.abs(df['SMA_Zscore']) > 3  # Flag as outlier if Z-score > 3

# Outlier Treatment (e.g., capping)
df.loc[df['Outlier_Flag'], 'SMA'] = df['SMA'].median()

# 7. Datetime Indexing
df.set_index('Datetime', inplace=True)

# 8. Differencing for Stationarity
df['SMA_Diff_1st'] = df['SMA'].diff()  # First-order differencing
df['SMA_Diff_Seasonal'] = df['SMA'].diff(24)  # Seasonal differencing (e.g., daily if hourly data)

# 9. Lag Features
# Creating lagged variables
df['SMA_Lag_1'] = df['SMA'].shift(1)
df['SMA_Lag_2'] = df['SMA'].shift(2)
df['SMA_Lag_3'] = df['SMA'].shift(3)

# Display the processed data
print(df.head())

# Save the preprocessed dataset to a new CSV file
df.to_csv('preprocessed_dataset.csv', index=True)
```

Feature Extraction

```
df = pd.read_csv('SMA_data.csv')
df.info()
import pandas as pd
import numpy as np
from scipy import signal
from scipy.fft import fft
from statsmodels.tsa.ar_model import AutoReg
from sklearn.preprocessing import PolynomialFeatures
```

```
df['Datetime'] = pd.to_datetime(df['Datetime'])
df.set_index('Datetime', inplace=True)

# Rolling Statistics (Window-Based Features)
df['rolling_mean'] = df['SMA'].rolling(window=3).mean()
df['rolling_std'] = df['SMA'].rolling(window=3).std()
df['rolling_min'] = df['SMA'].rolling(window=3).min()
df['rolling_max'] = df['SMA'].rolling(window=3).max()

# Linear Trend Feature
X = np.arange(len(df)).reshape(-1, 1)
poly = PolynomialFeatures(degree=1)
X_poly = poly.fit_transform(X)
df['linear_trend'] = X_poly[:, 1]

# Exponential Moving Average
df['ema'] = df['SMA'].ewm(span=5, adjust=False).mean()

# Frequency Domain Features (Fourier Transform)
fft_values = fft(df['SMA'].values)
df['fft_real'] = np.real(fft_values)
df['fft_imag'] = np.imag(fft_values)

# Lag-Based Features
df['lag_1'] = df['SMA'].shift(1)
df['lag_2'] = df['SMA'].shift(2)

# Autoregressive Features
model = AutoReg(df['SMA'].dropna(), lags=2)
ar_model = model.fit()
df['ar_coef_1'] = ar_model.params[1]  # First AR coefficient
df['ar_coef_2'] = ar_model.params[2]  # Second AR coefficient

# Cumulative Features
df['cumulative_sum'] = df['SMA'].cumsum()
df['cumulative_max'] = df['SMA'].cummax()
df['cumulative_min'] = df['SMA'].cummin()

# Differencing and Change Features
df['diff_1'] = df['SMA'].diff(1)
df['diff_2'] = df['SMA'].diff(2)
df['pct_change'] = df['SMA'].pct_change()
df['direction'] = np.sign(df['SMA'].diff())

# Anomaly/Change Detection Features
df['z_score'] = (df['SMA'] - df['SMA'].mean()) / df['SMA'].std()
df['outlier'] = np.abs(df['z_score']) > 2  # Mark points as outliers if Z-score > 2

# Time-Based Features
df['hour'] = df.index.hour
df['day_of_week'] = df.index.dayofweek
df['month'] = df.index.month

# Segment-Level Features (Segment by day)
df['date'] = df.index.date
daily_groups = df.groupby('date')['SMA']
df['daily_mean'] = daily_groups.transform('mean')
df['daily_std'] = daily_groups.transform('std')

# Drop temporary columns used for segmentation
df.drop(columns=['date'], inplace=True)

# Display the resulting DataFrame with extracted features
print(df.head())
```

previous code

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, MinMaxScaler, PolynomialFeatures
from scipy.fft import fft
from statsmodels.tsa.ar_model import AutoReg
from statsmodels.tsa.arima.model import ARIMA
from sklearn.ensemble import IsolationForest
from sklearn.svm import OneClassSVM
from sklearn.cluster import KMeans, DBSCAN
from ruptures import Pelt
from scipy.stats import zscore
```

```
import matplotlib.pyplot as plt

# --- Load and Preprocess Data --- #
df = pd.read_csv('SMA_data.csv', parse_dates=['Datetime'])

# Handling missing data
df['SMA'] = df['SMA'].interpolate(method='linear')

# Feature engineering: time-based features
df['Year'] = df['Datetime'].dt.year
df['Month'] = df['Datetime'].dt.month
df['Day'] = df['Datetime'].dt.day
df['Hour'] = df['Datetime'].dt.hour

# Time differences and SMA-based features
df['Time_Diff'] = df['Datetime'].diff().dt.total_seconds()
df['SMA_Diff'] = df['SMA'].diff()
df['SMA_Pct_Change'] = df['SMA'].pct_change()
df['SMA_CumSum'] = df['SMA_Diff'].cumsum()

# Data Normalization (Standardization and Min-Max scaling)
scaler_standard = StandardScaler()
df['SMA_Standardized'] = scaler_standard.fit_transform(df[['SMA']])

scaler_minmax = MinMaxScaler()
df['SMA_MinMax'] = scaler_minmax.fit_transform(df[['SMA']])

# Outlier detection and treatment using Z-score
df['SMA_Zscore'] = zscore(df['SMA'])
df['Outlier_Flag'] = np.abs(df['SMA_Zscore']) > 3
df.loc[df['Outlier_Flag'], 'SMA'] = df['SMA'].median()

# --- Feature Extraction --- #
# Rolling statistics
df['rolling_mean'] = df['SMA'].rolling(window=3).mean()
df['rolling_std'] = df['SMA'].rolling(window=3).std()
df['rolling_min'] = df['SMA'].rolling(window=3).min()
df['rolling_max'] = df['SMA'].rolling(window=3).max()

# Linear trend feature
X = np.arange(len(df)).reshape(-1, 1)
poly = PolynomialFeatures(degree=1)
X_poly = poly.fit_transform(X)
df['linear_trend'] = X_poly[:, 1]

# Exponential moving average (EMA)
df['ema'] = df['SMA'].ewm(span=5, adjust=False).mean()

# Fourier transform features
fft_values = fft(df['SMA'].values)
df['fft_real'] = np.real(fft_values)
df['fft_imag'] = np.imag(fft_values)

# Autoregressive (AR) features
ar_model = AutoReg(df['SMA'].dropna(), lags=2).fit()
df['ar_coef_1'] = ar_model.params[1]
df['ar_coef_2'] = ar_model.params[2]

# Cumulative features
df['cumulative_sum'] = df['SMA'].cumsum()
```

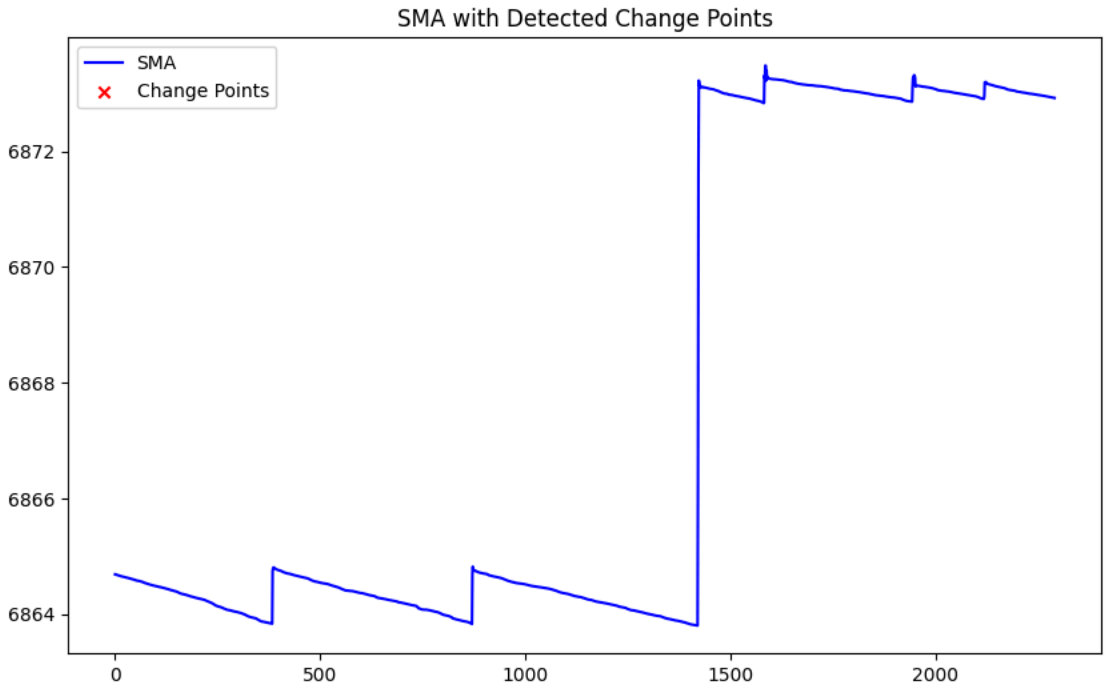
```
<ipython-input-5-d0f13f4b16e2>:67: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future ver:
  df['ar_coef_1'] = ar_model.params[1]
<ipython-input-5-d0f13f4b16e2>:68: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future ver:
  df['ar_coef_2'] = ar_model.params[2]
```

```
import numpy as np
import pandas as pd
import ruptures as rpt
from statsmodels.tsa.ar_model import AutoReg
from sklearn.ensemble import IsolationForest
from sklearn.svm import OneClassSVM

# 1. Change Point Detection (using ruptures)
def detect_change_points(series):
    # Fit the AutoReg model with lag 2
    ar_model = AutoReg(series, lags=2).fit()

    # Detect change points using ruptures
```

```
    algo = rpt.Pelt(model="ar").fit(series)
    change_points = algo.predict(pen=10)

    # Create a full-length array with NaN or 0s, marking change points with 1
    change_point_flags = np.zeros(len(series))
    for cp in change_points:
        if cp < len(change_point_flags):  # Ensure it's within bounds
            change_point_flags[cp] = 1  # Mark change points

    return change_point_flags

# Ensure the length of the change_points matches the length of the DataFrame
df['change_points'] = detect_change_points(df['SMA'].values)

# 2. Anomaly Detection (Isolation Forest & One-Class SVM)
# Fit Isolation Forest
iso_forest = IsolationForest(contamination=0.01)
df['anomaly_iforest'] = iso_forest.fit_predict(df[['SMA']])

# Fit One-Class SVM
svm = OneClassSVM(kernel='rbf', gamma=0.001, nu=0.05)
df['anomaly_svm'] = svm.fit_predict(df[['SMA']])

# 3. Output the updated DataFrame
print(df[['SMA', 'change_points', 'anomaly_iforest', 'anomaly_svm']].head())
```

```
           SMA  change_points  anomaly_iforest  anomaly_svm
0  6864.691463            0.0                1            1
1  6864.689664            0.0                1            1
2  6864.688585            0.0                1            1
3  6864.684927            0.0                1            1
4  6864.682858            0.0                1            1
```

```
import matplotlib.pyplot as plt

# Plot SMA with change points
plt.figure(figsize=(10, 6))
plt.plot(df['SMA'], label='SMA', color='blue')
plt.scatter(df.index[df['change_points'] == 1], df['SMA'][df['change_points'] == 1],
            color='red', label='Change Points', marker='x')
plt.title('SMA with Detected Change Points')
plt.legend()
plt.show()
```




```
# Adjust penalty for change point detection
change_points = detect_change_points(df['SMA'].values)

# Adjust contamination for Isolation Forest
iso_forest = IsolationForest(contamination=0.05)  # Try a higher contamination level
df['anomaly_iforest'] = iso_forest.fit_predict(df[['SMA']])
```