Auto-Correlation and SVM

```
import yfinance as yf
import pandas as pd
import numpy as np
from statsmodels.tsa.stattools import acf
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
```

```
ticker_symbol = "^NSEI"
data = yf.download(ticker_symbol, start="2015-01-01", end="2023-07-05")
data.fillna(method='ffill', inplace=True)
data = data[['Close', 'Volume']]
train_data = data[data.index < '2023-01-01']
test_data = data[data.index >= '2023-01-01']
```

```
[*********************100%%**********************]  1 of 1 completed
```

```
autocorr = acf(train_data['Close'], nlags=20)
print("Autocorrelation values:", autocorr)
def rolling_autocorr(series, window):
    return series.rolling(window).apply(lambda x: x.autocorr(), raw=False)
window_size = 20
train_data['Autocorr'] = rolling_autocorr(train_data['Close'], window=window_size)
test_data['Autocorr'] = rolling_autocorr(test_data['Close'], window=window_size)

train_data.dropna(inplace=True)
test_data.dropna(inplace=True)
```

Show hidden output

```
X_train = train_data[['Close', 'Volume', 'Autocorr']]
y_train = train_data['Close']
X_test = test_data[['Close', 'Volume', 'Autocorr']]
y_test = test_data['Close']
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
svm_model = SVR(kernel='rbf')
svm_model.fit(X_train_scaled, y_train)
```

```
y_pred = svm_model.predict(X_test_scaled)

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
comparison_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred}, index=test_data.index)
print(comparison_df.head())
```

Show hidden output

Introducing BB, Moving average, EMA

```
data['MA_200'] = data['Close'].rolling(window=200).mean()
data['MA_50'] = data['Close'].rolling(window=50).mean()
data['MA_20'] = data['Close'].rolling(window=20).mean()

# Calculate the Bollinger Bands (20-day MA, 2 standard deviations)
data['BB_upper'] = data['MA_20'] + (data['Close'].rolling(window=20).std() * 2)
data['BB_lower'] = data['MA_20'] - (data['Close'].rolling(window=20).std() * 2)

# Calculate the Exponential Moving Average (20-day EMA)
data['EMA_20'] = data['Close'].ewm(span=20, adjust=False).mean()
```

Show hidden output

```
data.fillna(method='ffill', inplace=True)
data.dropna(inplace=True)
train_data = data[data.index < '2023-01-01']
test_data = data[data.index >= '2023-01-01']

def rolling_autocorr(series, window):
    return series.rolling(window).apply(lambda x: x.autocorr(), raw=False)
window_size = 20

train_data['Autocorr'] = rolling_autocorr(train_data['Close'], window=window_size)
test_data['Autocorr'] = rolling_autocorr(test_data['Close'], window=window_size)

train_data.dropna(inplace=True)
test_data.dropna(inplace=True)
```

Show hidden output

```
features = ['Close', 'Volume', 'Autocorr', 'MA_200', 'MA_50', 'MA_20', 'BB_upper', 'BB_lower', 'EMA_20']
X_train = train_data[features]
y_train = train_data['Close']
X_test = test_data[features]
y_test = test_data['Close']

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

svm_model = SVR(kernel='rbf')
svm_model.fit(X_train_scaled, y_train)

y_pred = svm_model.predict(X_test_scaled)

# Calculate the Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 41993485.79888248
```

Lag feature and Techinal Indicator

```
!pip install ta
```

Show hidden output

```
import ta
data['RSI'] = ta.momentum.RSIIndicator(close=data['Close'], window=14).rsi()
macd = ta.trend.MACD(close=data['Close'])
data['MACD'] = macd.macd()
data['MACD_Signal'] = macd.macd_signal()
data['MACD_Diff'] = macd.macd_diff()
for lag in range(1, 6):
    data[f'Close_Lag_{lag}'] = data['Close'].shift(lag)
data.dropna(inplace=True)
```

```
data.fillna(method='ffill', inplace=True)
data.dropna(inplace=True)
train_data = data[data.index < '2023-01-01']
test_data = data[data.index >= '2023-01-01']

def rolling_autocorr(series, window):
    return series.rolling(window).apply(lambda x: x.autocorr(), raw=False)

window_size = 20
```

```
train_data['Autocorr'] = rolling_autocorr(train_data['Close'], window=window_size)
test_data['Autocorr'] = rolling_autocorr(test_data['Close'], window=window_size)
train_data.dropna(inplace=True)
test_data.dropna(inplace=True)
```

Show hidden output

```
features = ['Close', 'Volume', 'Autocorr', 'MA_200', 'MA_50', 'MA_20', 'BB_upper', 'BB_lower', 'EMA_20',
            'RSI', 'MACD', 'MACD_Signal', 'MACD_Diff', 'Close_Lag_1', 'Close_Lag_2', 'Close_Lag_3', 'Close_Lag_4', 'Close_Lag_5']
X_train = train_data[features]
y_train = train_data['Close']
X_test = test_data[features]
y_test = test_data['Close']
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

svm_model = SVR(kernel='rbf')
svm_model.fit(X_train_scaled, y_train)
y_pred = svm_model.predict(X_test_scaled)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 42498808.37773062
```

Grid SearchCV and Gradient Booster

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
features = ['Close', 'Volume', 'MA_200', 'MA_50', 'MA_20', 'BB_upper', 'BB_lower', 'EMA_20',
            'RSI', 'MACD', 'MACD_Signal', 'MACD_Diff', 'Close_Lag_1', 'Close_Lag_2', 'Close_Lag_3', 'Close_Lag_4', 'Close_Lag_5']
X_train = train_data[features]
y_train = train_data['Close']
X_test = test_data[features]
y_test = test_data['Close']

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
gb_model = GradientBoostingRegressor()
param_grid = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 3]
}
grid_search = GridSearchCV(estimator=gb_model, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)
grid_search.fit(X_train_scaled, y_train)


best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_scaled)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
Mean Squared Error: 9405.327238594426
```

```
data['Fibonacci_21'] = data['Close'].rolling(window=21).mean()
data['Fibonacci_34'] = data['Close'].rolling(window=34).mean()
data['Fibonacci_55'] = data['Close'].rolling(window=55).mean()


def hull_moving_average(data, window):
    half_window = int(window / 2)
    sqrt_window = int(np.sqrt(window))
    wma_half = 2 * data.rolling(window=half_window).mean()
    wma_full = data.rolling(window=window).mean()
    hma = (wma_half - wma_full).rolling(window=sqrt_window).mean()
    return hma

data['HMA_20'] = hull_moving_average(data['Close'], 20)
```

```
for lag in range(1, 6):
    data[f'Close_Lag_{lag}'] = data['Close'].shift(lag)
data.dropna(inplace=True)
train_data = data[data.index < '2023-01-01']
test_data = data[data.index >= '2023-01-01']


features = ['Close', 'Volume', 'MA_200', 'MA_50', 'MA_20', 'BB_upper', 'BB_lower', 'EMA_20',
            'RSI', 'MACD', 'MACD_Signal', 'MACD_Diff', 'Close_Lag_1', 'Close_Lag_2', 'Close_Lag_3',
            'Close_Lag_4', 'Close_Lag_5', 'Fibonacci_21', 'Fibonacci_34', 'Fibonacci_55', 'HMA_20']
X_train = train_data[features]
y_train = train_data['Close']
X_test = test_data[features]
y_test = test_data['Close']

# Standardize the feature
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize Gradient Booster
gb_model = GradientBoostingRegressor()
param_grid = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 3]
}
grid_search = GridSearchCV(estimator=gb_model, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)
grid_search.fit(X_train_scaled, y_train)
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_scaled)

# Calculate the Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
Mean Squared Error: 7892.258722360177
```

Start coding or generate with AI