In [25]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC, SVR
from sklearn.metrics import accuracy_score, mean_squared_error
import matplotlib.pyplot as plt
```

In [26]:
```
# Load dataset
file_path = 'NFLX.csv'
data = pd.read_csv(file_path)
```

In [27]:
```
# Display the first few rows of the dataset
print(data.head())
```

```
         Date        Open        High         Low       Close   Adj Close  \
0  2018-02-05  262.000000  267.899994  250.029999  254.259995  254.259995
1  2018-02-06  247.699997  266.700012  245.000000  265.720001  265.720001
2  2018-02-07  266.579987  272.450012  264.329987  264.559998  264.559998
3  2018-02-08  267.079987  267.619995  250.000000  250.100006  250.100006
4  2018-02-09  253.850006  255.800003  236.110001  249.470001  249.470001

     Volume
0  11896100
1  12595800
2   8981500
3   9306700
4  16906900
```

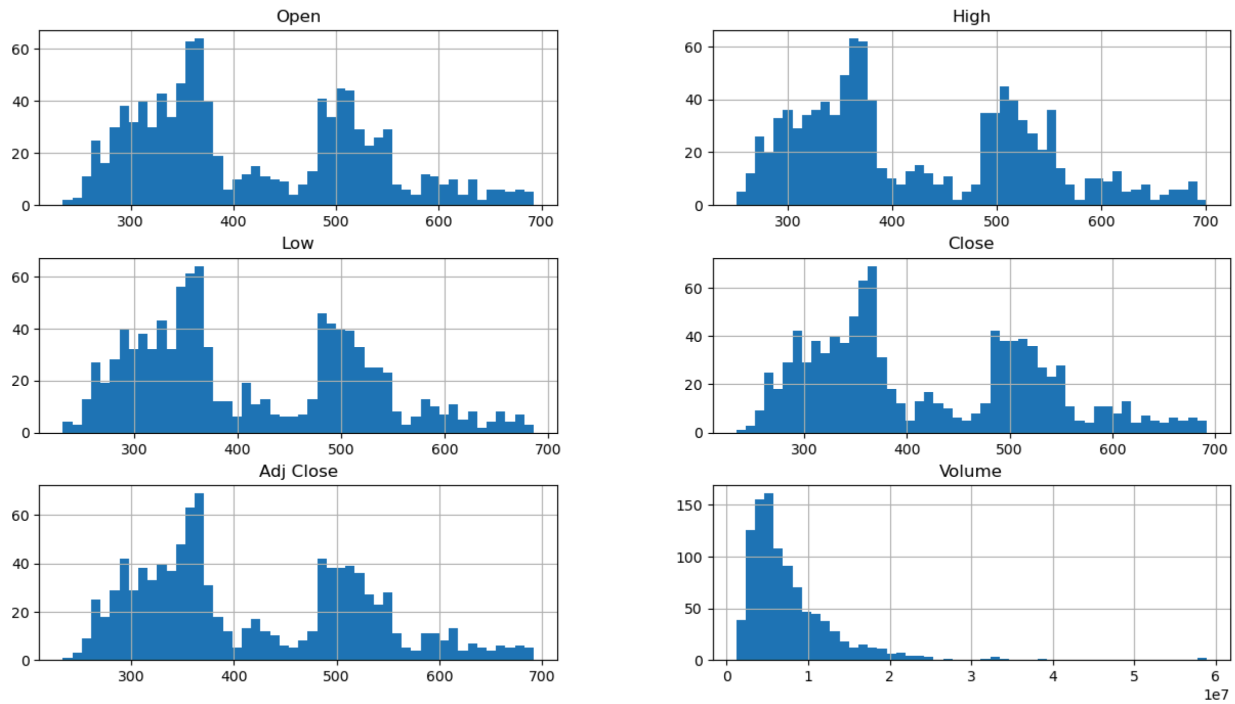
In [28]:
```
# Looking a the basic statistics about the data

display(data.describe())
```

	Open	High	Low	Close	Adj Close	Volume
count	1009.000000	1009.000000	1009.000000	1009.000000	1009.000000	1.009000e+03
mean	419.059673	425.320703	412.374044	419.000733	419.000733	7.570685e+06
std	108.537532	109.262960	107.555867	108.289999	108.289999	5.465535e+06
min	233.919998	250.649994	231.229996	233.880005	233.880005	1.144000e+06
25%	331.489990	336.299988	326.000000	331.619995	331.619995	4.091900e+06
50%	377.769989	383.010010	370.880005	378.670013	378.670013	5.934500e+06
75%	509.130005	515.630005	502.529999	509.079987	509.079987	9.322400e+06
max	692.349976	700.989990	686.090027	691.690002	691.690002	5.890430e+07

In [29]:
```
# Looking at the data distributions

data.hist(bins=50, figsize=(15,8))
plt.show()
```



In [30]:
```
# Check for duplicate observations
duplicate_count = data.duplicated().sum()
print(f"There are {duplicate_count} duplicates in the dataset")

# Delete duplicate data if found
if duplicate_count > 0:
    data.drop_duplicates#(inplace=True)
    print("\nDuplicate observations were deleted.")
    # Print the shape of data after removal of duplicates
    print("Data shape after duplicate removal:{0}".format(data.shape))
```

```
There are 0 duplicates in the dataset
```

In [31]:
```
# Gets number of unique values for each column
unique_values_per_attrib = data.nunique()

# Records columns to delete
single_value_columns = [i for i, value_count in enumerate(unique_values_per_attrib) if
print("There are", len(single_value_columns), "single-valued columns in the datsset")

# Deletes single-value columns, if exist
if len(single_value_columns) > 0:
    data.drop(single_value_columns)#, axis=1, inplace=True)
    print("\n\tSingle-valued columns were removed.")
    # Prints the shape of data after removal of single-value columns
    print("\n\tData shape after single-value column removal:", data.shape)
```

```
There are 0 single-valued columns in the datsset
```

In [32]:
```
# Handle missing values
data = data.dropna()
```

In [33]:
```
# Convert 'Date' to datetime format
data['Date'] = pd.to_datetime(data['Date'])
```

In [34]:
```
# Sort by date
data = data.sort_values(by='Date')
```

In [35]:
```python
# Use 'Close' prices for prediction
X = data[['Close']].values
y = np.sign(np.diff(X.flatten()))  # +1 for up, -1 for down, 0 for no change
y = np.concatenate(([0], y))  # Align y with X
```

In [36]:
```python
# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [37]:
```python
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random
```

In [38]:
```python
# Linear SVM Classification
linear_svm = SVC(kernel='linear')
linear_svm.fit(X_train, y_train)
y_pred_linear = linear_svm.predict(X_test)
print("Accuracy (Linear SVM):", accuracy_score(y_test, y_pred_linear))
```

```
Accuracy (Linear SVM): 0.44554455445544555
```

In [39]:
```python
# Soft Margin Classification
soft_margin_svm = SVC(kernel='linear', C=0.1)
soft_margin_svm.fit(X_train, y_train)
y_pred_soft = soft_margin_svm.predict(X_test)
print("Accuracy (Soft Margin SVM):", accuracy_score(y_test, y_pred_soft))
```

```
Accuracy (Soft Margin SVM): 0.44554455445544555
```

In [40]:
```python
# Nonlinear SVM Classification with Polynomial Kernel
poly_svm = SVC(kernel='poly', degree=3, C=1)
poly_svm.fit(X_train, y_train)
y_pred_poly = poly_svm.predict(X_test)
print("Accuracy (Polynomial Kernel SVM):", accuracy_score(y_test, y_pred_poly))
```

```
Accuracy (Polynomial Kernel SVM): 0.44554455445544555
```

In [41]:
```python
# Nonlinear SVM Classification with Gaussian RBF Kernel
rbf_svm = SVC(kernel='rbf', gamma='scale', C=1)
rbf_svm.fit(X_train, y_train)
y_pred_rbf = rbf_svm.predict(X_test)
print("Accuracy (RBF Kernel SVM):", accuracy_score(y_test, y_pred_rbf))
```

```
Accuracy (RBF Kernel SVM): 0.5148514851485149
```

In [42]:
```python
# Preparing data for SVR
X_prices = X_scaled[:-1]  # Use all but the last day
y_prices = X_scaled[1:, 0]  # Predict the closing price for the next day
```

In [43]:
```python
# Hyperparameter tuning for SVR
svr_param_grid = {
    'kernel': ['linear', 'poly', 'rbf'],
    'C': [0.1, 1, 10],
    'degree': [2, 3, 4],
    'gamma': ['scale', 'auto'],
    'epsilon': [0.01, 0.1, 0.2]
}
best_svr = GridSearchCV(SVR(), svr_param_grid, cv=5, n_jobs=-1, scoring='neg_mean_squa
```

```
best_svr.fit(X_prices, y_prices)
best_svr_model = best_svr.best_estimator_
```

In [44]:
```
# SVM Regression for Predicting Prices
best_svr_model.fit(X_prices, y_prices)
```

Out[44]:

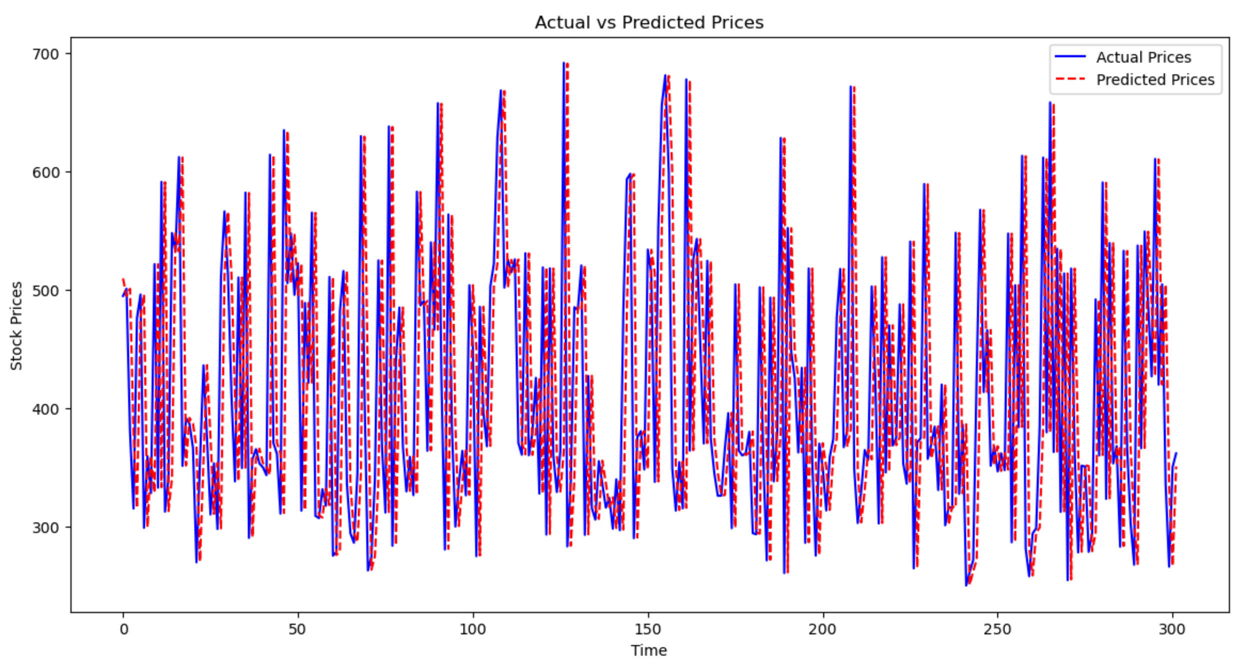
▾ SVR

```
SVR(C=10, degree=2, epsilon=0.01, kernel='linear')
```

In [45]:
```
# Predicting the test set
predicted_prices = best_svr_model.predict(X_test[:-1])
mse = mean_squared_error(X_test[1:, 0], predicted_prices)
print("MSE (Best SVR):", mse)
```

```
MSE (Best SVR): 1.9008962177167037
```

In [46]:
```
# Visualizing Actual vs Predicted Prices
plt.figure(figsize=(14, 7))
plt.plot(scaler.inverse_transform(X_test[1:])[:, 0], color='blue', label='Actual Price
plt.plot(scaler.inverse_transform(np.hstack([predicted_prices.reshape(-1, 1), np.zeros
plt.title('Actual vs Predicted Prices')
plt.xlabel('Time')
plt.ylabel('Stock Prices')
plt.legend()
plt.show()
```



In [ ]:

In [ ]: