Seminar 11:
ML Application:
Stock Price Trend Prediction
Fraud Detection

Readings:

1. McKinsey Report - A new approach to fighting fraud while enhancing customer experience

Agenda

- Stock Price Prediction
- Challenge of Fraud Detection & Solutions with AI Techniques for Fraud Detection

Predicting stock price trend

Installation & Download Data

- https://github.com/dataquestio/project-walkthroughs/tree/master/sp_500
- https://www.youtube.com/watch?v=1O_BenficgE
- To implement this project, please install the following locally:
  - JupyerLab
  - Python 3.8+
  - Python packages
    - pandas
    - yfinance
    - scikit-learn

Importing Libraries

Importing necessary libraries:

- yfinance: Used for fetching financial data.
- pandas: Essential for data manipulation and analysis.
- os: For interacting with the operating system, particularly for file operations.

In [4]:

```
import yfinance as yf
import pandas as pd
import os
```

Data Acquisition

- The code checks if a file named "sp500.csv" exists. If it does, it loads the data from this file.
- If the file doesn't exist, it uses yfinance to fetch the historical data of the S&P 500 index (denoted as "^GSPC") and saves this data to "sp500.csv" for future use. This step ensures efficiency by reducing the need to fetch data from the internet every time.

In [5]:

```
if os.path.exists("sp500.csv"):
    sp500 = pd.read_csv("sp500.csv", index_col=0)
else:
    sp500 = yf.Ticker("^GSPC")
    sp500 = sp500.history(period="max")
    sp500.to_csv("sp500.csv")
```

Data Visualization:

- The index of the sp500 DataFrame is converted to datetime format to better handle time series data.
  - This step is crucial for time series analysis as it facilitates date-wise data manipulation and visualization.

```
In [6]: sp500.index = pd.to_datetime(sp500.index)

In [7]: sp500
```

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
1927-12-30 00:00:00-05:00	17.660000	17.660000	17.660000	17.660000	0	0.0	0.0
1928-01-03 00:00:00-05:00	17.760000	17.760000	17.760000	17.760000	0	0.0	0.0
1928-01-04 00:00:00-05:00	17.719999	17.719999	17.719999	17.719999	0	0.0	0.0
1928-01-05 00:00:00-05:00	17.549999	17.549999	17.549999	17.549999	0	0.0	0.0
1928-01-06 00:00:00-05:00	17.660000	17.660000	17.660000	17.660000	0	0.0	0.0
...	...	...	...	...	...	...	...
2023-12-22 00:00:00-05:00	4753.919922	4772.939941	4736.770020	4754.629883	3046770000	0.0	0.0
2023-12-26 00:00:00-05:00	4758.859863	4784.720215	4758.450195	4774.750000	2513910000	0.0	0.0
2023-12-27 00:00:00-05:00	4773.450195	4785.390137	4768.899902	4781.580078	2748450000	0.0	0.0
2023-12-28 00:00:00-05:00	4786.439941	4793.299805	4780.979980	4783.350098	2698860000	0.0	0.0
2023-12-29 00:00:00-05:00	4782.879883	4788.430176	4751.990234	4769.830078	3126060000	0.0	0.0

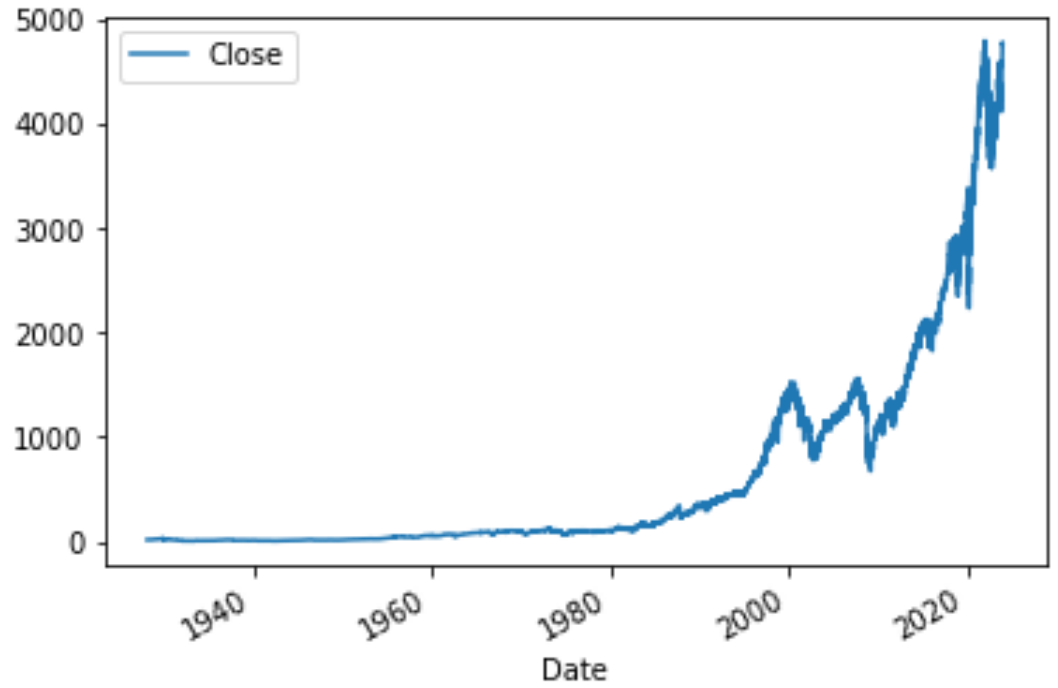
24115 rows × 7 columns

Data Visualization:

- A line plot is created for the "Close" prices of the S&P 500 index over time.
- This visual representation is important to understand the overall trend and patterns in the stock prices.

```
In [8]: sp500.plot.line(y="Close", use_index=True)

Out[8]: <AxesSubplot:xlabel='Date'>
```



Data Preparation:

- **Column Removal:** Unnecessary data columns "Dividends" and "Stock Splits" are removed from the S&P 500 dataset.

In [9]:
```
del sp500["Dividends"]
del sp500["Stock Splits"]
```

- **Label:** A new column "Tomorrow" is created to hold the next day's closing prices, and a target variable "Target" is generated to indicate whether the price will go up (1) or not (0) compared to the current day's closing price.

In [10]:
```
sp500["Tomorrow"] = sp500["Close"].shift(-1)
```

In [11]:
```
sp500["Target"] = (sp500["Tomorrow"] > sp500["Close"]).astype(int)
```

- **Data Filtering:** The dataset is filtered to include records starting from January 1, 1990 only.

```
sp500 = sp500[sp500.index >= pd.to_datetime("1990-01-01", utc=True)].copy()
```

Model Initialization:

- **Importing Model:** The RandomForestClassifier from the scikit-learn library is imported for use.
- **Configuring Model:** The RandomForest model is initialized with specific parameters to control the number of trees and the complexity of the trees.

In [41]:

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100, min_samples_split=100, random_state=1)
```

Training and Testing:

- **Data Splitting:** The dataset is split into a training set (all data except the last 100 entries) and a test set (the last 100 entries).
- **Model Training:** The model is trained on the training set using selected features such as closing price and trading volume to learn patterns associated with price increases.
- **Model Output:** The trained model's configuration is outputted, confirming the setup of the machine learning model.

```
train = sp500.iloc[:-100]
test = sp500.iloc[-100:]

predictors = ["Close", "Volume", "Open", "High", "Low"]
model.fit(train[predictors], train["Target"])
```

```
Out[41]: RandomForestClassifier(min_samples_split=100, random_state=1)
```

Model Prediction and Evaluation:

- **Generating Predictions:** The RandomForest model predicts whether the stock prices will go up using the test dataset.
- **Evaluating Model Precision:** The precision_score function from scikit-learn calculates the precision of the model, which is the ratio of correctly predicted positive observations to the total predicted positives.

In [42]:
```
from sklearn.metrics import precision_score

preds = model.predict(test[predictors])
preds = pd.Series(preds, index=test.index)
precision_score(test["Target"], preds)
```

Out[42]: 0.47058823529411764

Evaluation Metrics

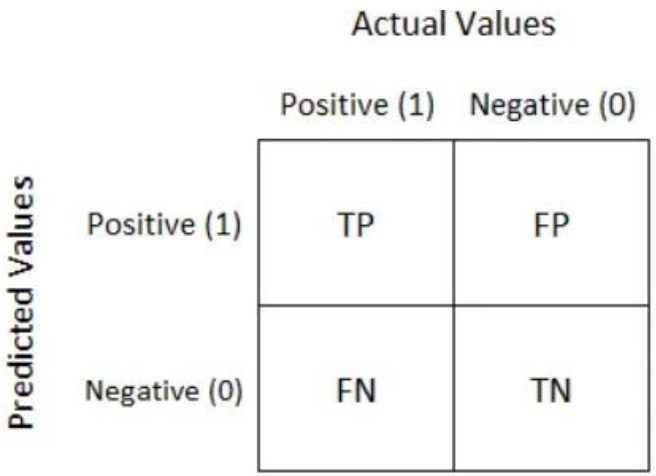
- Recap- Regression Metrics
  - Root Mean Square Error (RMSE)

$$\text{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2}$$

  - Mean Square Error

Evaluation Metrics: Classification

- Confusion matrix: a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class.



Evaluation Metrics: Classification

- Accuracy: the number of test cases correctly classified divided by the total number of test cases
  - Provide a good holistic view
  - Answers the questions “Out of all the predictions our model made, what percentage were correct?”
  - When it comes to unbalanced datasets, you can have high accuracy and still have a completely ineffective model.

$$Accuracy = (TP+TN)/(TP+TN+FP+FN)$$

Evaluation Metrics: Classification

- Precision: the metric used to identify the correctness of classification
  - The greater the fraction, the higher is the precision, which means the better the ability of the model to correctly classify the positive class.
  - Answers the questions "Out of all the times the model said an observation belonged to a class, how many times did it actually?"
  - A high precision score can be misleading because it tells nothing about how many times the model predicted false but it was actually true. Precision is a good measure when the costs of False Positive is high.

$$Precision = TP/(TP + FP)$$

Evaluation Metrics: Classification

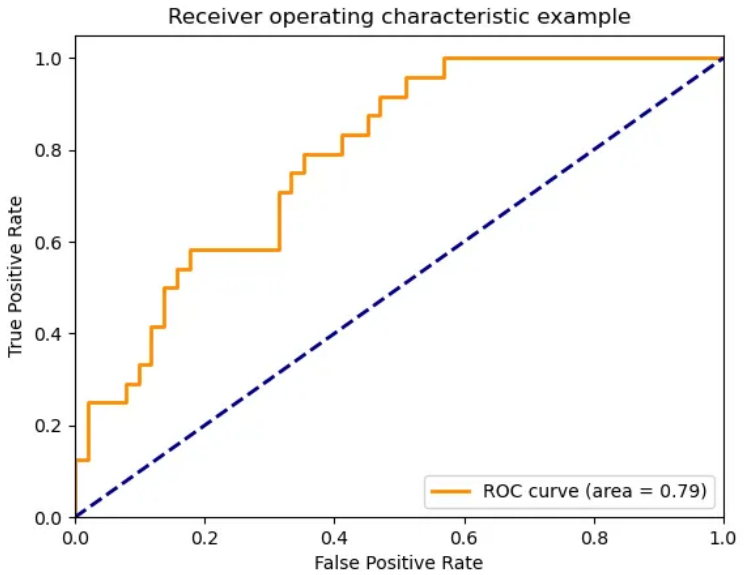
- ROC curve – receiver operating characteristic curve: a two-dimensional curve with the True Positive Rate on the vertical axis and False Positive Rate on the horizontal axis.
  - True Positive Rate

$$TPR = \frac{TP}{TP + FN}$$

  - False positive Rate

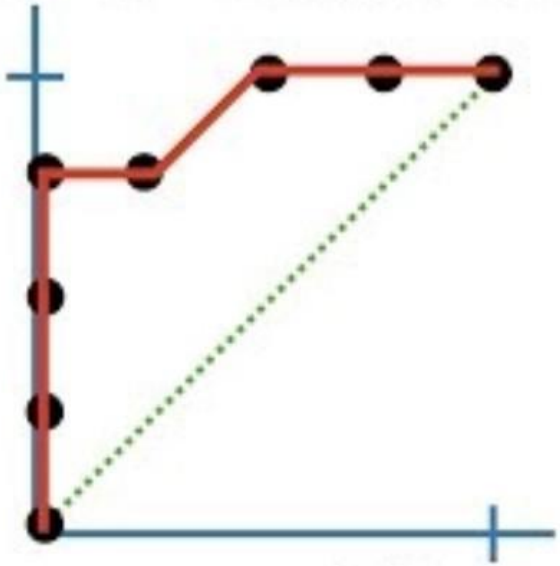
$$FPR = \frac{FP}{FP + TN}$$

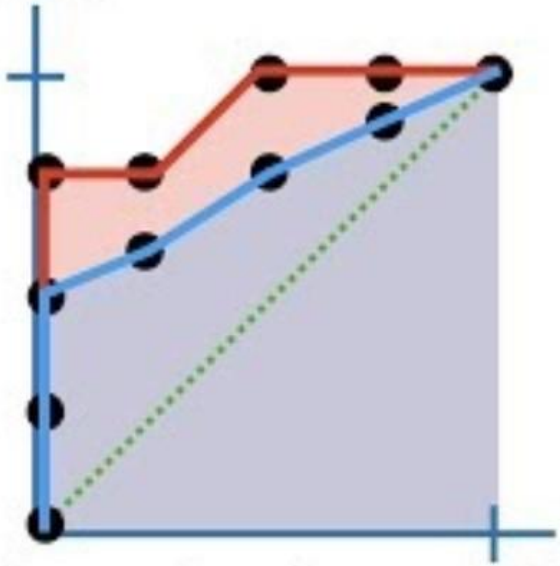
- The AUC (Area Under Curve) indicates the area of the ROC curve.



Source: https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html#sphx-glr-auto-examples-model-selection-plot-roc-py

ROC and AUC....





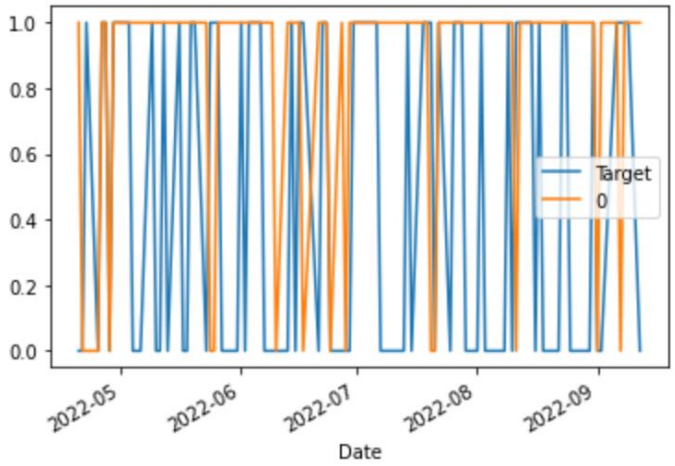
...Clearly Explained!!!

Combining and Visualizing Results:

Plotting Results: A plot is generated to visually compare the predictions with the actual target values across the dates in the test set.

In [43]:
```
combined = pd.concat([test["Target"], preds], axis=1)
combined.plot()
```

Out[43]: <AxesSubplot:xlabel='Date'>



predict Function:

- Train the model with the given training data and predictors.
- Creates a pandas Series of predictions with appropriate indexing.
- Combines predictions with the actual target values for comparison.

In [44]:

```
def predict(train, test, predictors, model):
    model.fit(train[predictors], train["Target"])
    preds = model.predict(test[predictors])
    preds = pd.Series(preds, index=test.index, name="Predictions")
    combined = pd.concat([test["Target"], preds], axis=1)
    return combined
```

backtest Function:

- Perform a rolling window backtest on the data with a specified starting point and step size.
- Iteratively split the data into training and test sets, apply the predict function, and append predictions.

In [45]:
```
def backtest(data, model, predictors, start=2500, step=250):
    all_predictions = []

    for i in range(start, data.shape[0], step):
        train = data.iloc[0:i].copy()
        test = data.iloc[i:(i+step)].copy()
        predictions = predict(train, test, predictors, model)
        all_predictions.append(predictions)

    return pd.concat(all_predictions)
```

In [46]:
```
predictions = backtest(sp500, model, predictors)
```

In [47]:
```
predictions["Predictions"].value_counts()
```

Out[47]:
```
0    3337
1    2401
Name: Predictions, dtype: int64
```

In [48]:
```
precision_score(predictions["Target"], predictions["Predictions"])
```

Out[48]:
```
0.534777176176593
```

In [49]:
```
predictions["Target"].value_counts() / predictions.shape[0]
```

Out[49]:
```
1    0.536075
0    0.463925
Name: Target, dtype: float64
```

Feature Engineering:

- **Creating Time Horizons:** New features based on rolling averages and trends for different time horizons (2, 5, 60, 250, 1000 days) are calculated.
- **Calculating Ratios:** For each horizon, a ratio of the current close price to the rolling average ('Close_Ratio') is computed.
- **Determining Trends:** A trend feature ('Trend') is also created by summing up the 'Target' values over each horizon, which indicates the general direction of stock movement in that period.
- **Finalizing Features:** These new features are then added to the list of predictors for the model.

In [50]:
```
horizons = [2,5,60,250,1000]
new_predictors = []

for horizon in horizons:
    rolling_averages = sp500.rolling(horizon).mean()

    ratio_column = f"Close_Ratio_{horizon}"
    sp500[ratio_column] = sp500["Close"] / rolling_averages["Close"]

    trend_column = f"Trend_{horizon}"
    sp500[trend_column] = sp500.shift(1).rolling(horizon).sum()["Target"]

    new_predictors+= [ratio_column, trend_column]
```

In [53]:
```
sp500 = sp500.dropna(subset=sp500.columns[sp500.columns != "Tomorrow"])
```

In [55]:
```
sp500
```

Out[55]:

Date	Open	High	Low	Close	Volume	Tomorrow	Target	Close_Ratio_2	Trend_2	Close_Ratio_5	Tren
1993-12-14	465.730011	466.119995	462.459991	463.059998	275050000	461.839996	0	0.997157	1.0	0.996617	
1993-12-15	463.059998	463.690002	461.839996	461.839996	331770000	463.339996	1	0.998681	0.0	0.995899	
1993-12-16	461.859985	463.980011	461.859985	463.339996	284620000	466.380005	1	1.001621	1.0	0.999495	
1993-12-17	463.339996	466.380005	463.339996	466.380005	363750000	465.850006	0	1.003270	2.0	1.004991	
1993-12-20	466.380005	466.899994	465.529999	465.850006	255900000	465.299988	0	0.999431	1.0	1.003784	

Model Update and Prediction:

- **Updating Model Parameters:** The RandomForestClassifier's parameters are adjusted to potentially improve its performance.
- **Adjusting Prediction Threshold:** The predict function is modified to use probability thresholds to decide on class predictions. If the probability of class 1 is above 0.6, it is predicted as 1; otherwise, it is predicted as 0.
- **Backtesting with New Predictors:** A backtesting function is applied using the new predictors to simulate the model's performance over the historical data.
- **Model Evaluation:** The precision score is calculated for the new model, along with the distribution of the predicted and actual target values, to assess the updated model's performance.

```
In [56]: model = RandomForestClassifier(n_estimators=200, min_samples_split=50, random_state=1)

In [57]: def predict(train, test, predictors, model):
             model.fit(train[predictors], train["Target"])
             preds = model.predict_proba(test[predictors])[:,1]
             preds[preds >=.6] = 1
             preds[preds <.6] = 0
             preds = pd.Series(preds, index=test.index, name="Predictions")
             combined = pd.concat([test["Target"], preds], axis=1)
             return combined

In [58]: predictions = backtest(sp500, model, new_predictors)

In [59]: predictions["Predictions"].value_counts()

Out[59]: 0.0    3933
         1.0     805
         Name: Predictions, dtype: int64

In [60]: precision_score(predictions["Target"], predictions["Predictions"])

Out[60]: 0.5701863354037268
```

How to improve?

Challenge of Fraud Detection

Fraud Types Detected by AI

Payment Card Fraud

Account Takeover (Ato) Fraud

Phishing and Cyberattacks

Insurance Fraud

Healthcare Fraud

E-Commerce Fraud

Application Fraud

Identity Theft

Money Laundering

Rising Fraud Losses

436%

Fraud losses in the United States have surged by 436% to $5.9 billion in 2021 compared to 2017.

392%

Internet crime losses have also surged by 392% to $6.9 billion, based on data from the Federal Trade Commission and the Federal Bureau of Investigation.

Alarming Figures

These alarming figures highlight the urgent need for effective fraud management strategies.

Persistent Fraud Attempts

74%

According to the 2021 AFP Payments Fraud and Control Survey, 74% of organizations have fallen victim to payment fraud attempts.

$32 Billion

Credit card fraud losses reached a staggering $32 billion globally in 2020, with the US accounting for over a third of these losses, as reported by Juniper Research.

Continued Challenge

The challenge of payment fraud attempts continues to persist and calls for proactive fraud management measures.

Impact on Customer Trust

Financial Losses

Fraud not only results in financial losses but also damages customers' trust and willingness to use digital services.

10% of Users

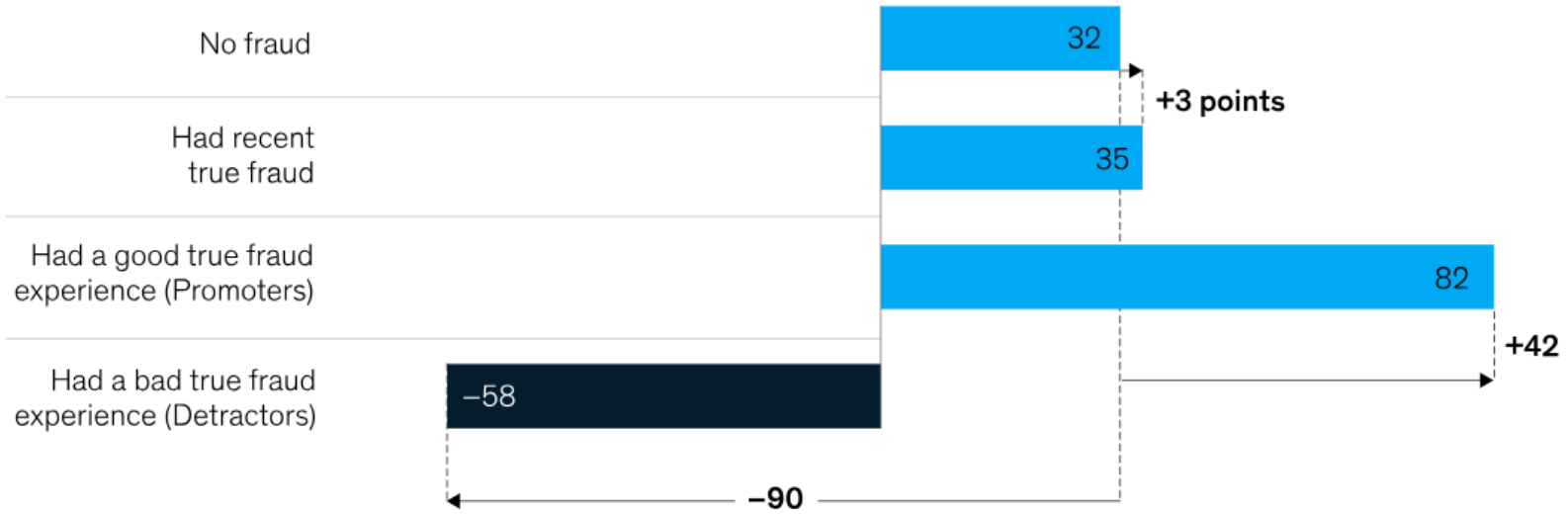
Over 10% of credit and debit card users have experienced fraud within a year, causing stress and preventing smooth transactions.

Customer Experience

A survey by McKinsey revealed that 70% of banking customers who were fraud victims reported feeling anxious, stressed, displeased, or frustrated when warned about potential fraud.

When companies respond well to fraud events, customers report higher levels of satisfaction.

Average customer satisfaction score for different customer groups, illustrative



McKinsey & Company

Advances in technology present challenges

- Attacks occur with frequency, speed, and effectiveness

Phishing

Social Engineering

Deep Fakes

Shifting to a Proactive Approach

1 Transitioning from Reactive to Proactive

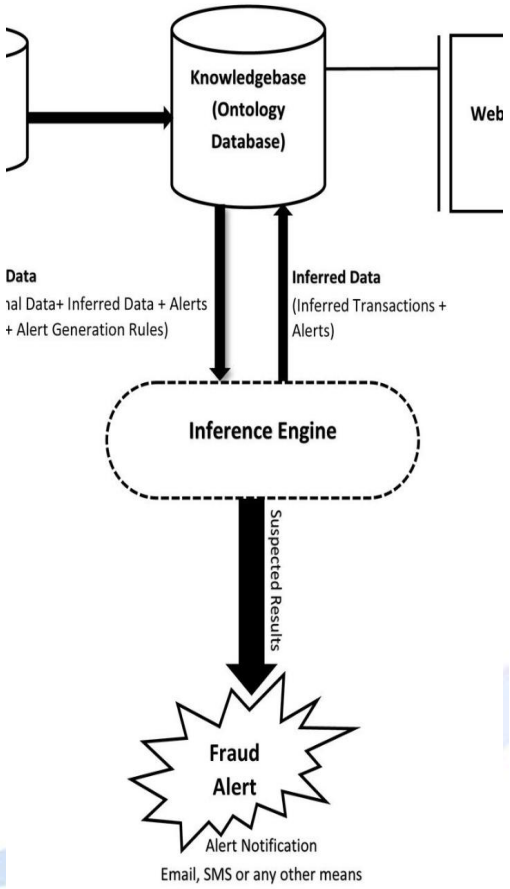
To combat fraud effectively, organizations must transition from reactive and siloed fraud mitigation to a proactive, customer-centric, integrated, and continuously evolving approach.

Utilizing AI/ML and Actionable Analytics 2

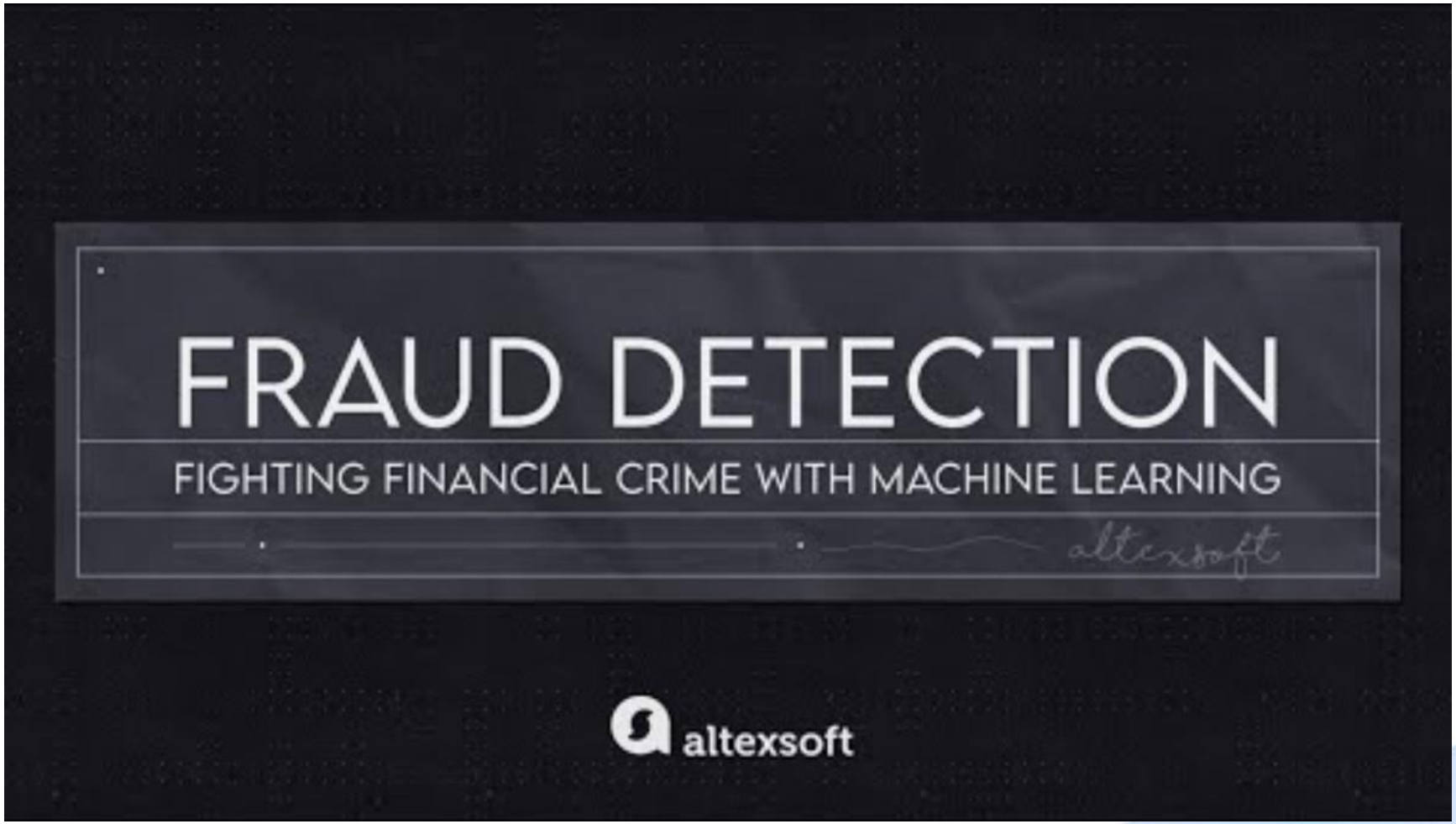
This entails relying more on AI/ML, employing actionable analytics, and leveraging technology to enhance customer experience and advanced authentication.

3 Embracing Technological Innovations

By embracing these advancements, organizations can stay one step ahead of fraudsters and protect their customers and business from potential threats.



Solutions with AI Techniques


FRAUD DETECTION
FIGHTING FINANCIAL CRIME WITH MACHINE LEARNING
altexsoft
altexsoft

Method

- Rule-based systems

Operate on predefined sets of rules that flag transactions or activities based on specific criteria.

Typically established by experts who have domain knowledge about common fraud patterns. For instance, the system might trigger an alert if a credit card transaction goes beyond a certain monetary threshold or occurs in a different country from the cardholder's location.

Limitation: While rule-based systems can swiftly identify well-known fraud scenarios, they struggle to detect novel or sophisticated fraud schemes. As fraudsters continuously develop new tactics to evade detection, rule-based systems can become outdated and ineffective unless regularly updated, leading to a cat-and-mouse chase between the defenders and the attackers.

Method

- Anomaly detection methods

Anomaly detection, also known as outlier detection, focuses on identifying instances that significantly deviate from the expected patterns within a dataset.

This method is particularly useful for uncovering unknown fraud patterns that rule-based systems might not cover. Statistical techniques, such as clustering and outlier analysis, are used to identify unusual behaviors that might indicate fraud.

Face challenges in distinguishing between genuine anomalies and legitimate variations in user behavior. False positives can overwhelm investigators and diminish the system's credibility, wasting valuable resources on investigating non-fraudulent cases. Moreover, these methods might struggle with detecting subtle anomalies in high-dimensional data, such as financial transactions.

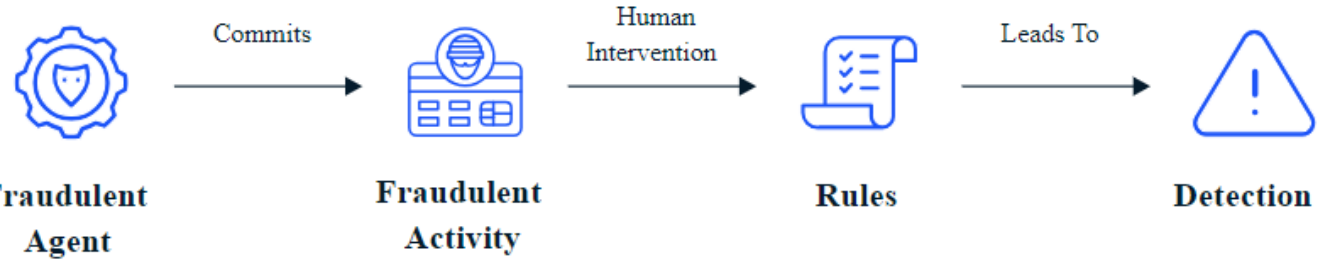
Limitations of traditional approaches

- Static and inflexible
  - Rule-based systems lack the flexibility to adapt to emerging fraud patterns, and anomaly detection methods struggle to keep up with novel tactics.
- High false positive rates
  - Both can generate numerous false positives, inundating investigators with irrelevant alerts and potentially missing genuine cases in the noise.
- Data complexity
  - Traditional methods might struggle to handle complex, high-dimensional data, making them less effective in identifying subtle fraud patterns.

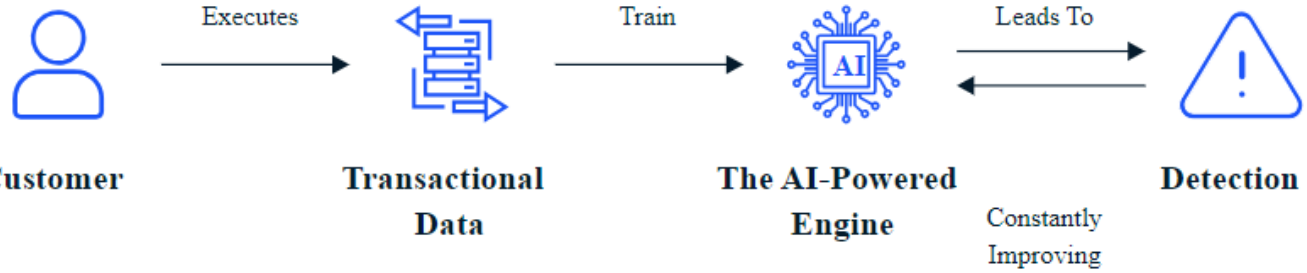
Limitations of traditional approaches

- Scalability
  - As transaction volumes increase, traditional methods might struggle to process and analyze vast amounts of data in real-time.
- Lack of contextual understanding
  - These methods often lack the ability to understand the context surrounding transactions, potentially resulting in misinterpretation and triggering false alarms.

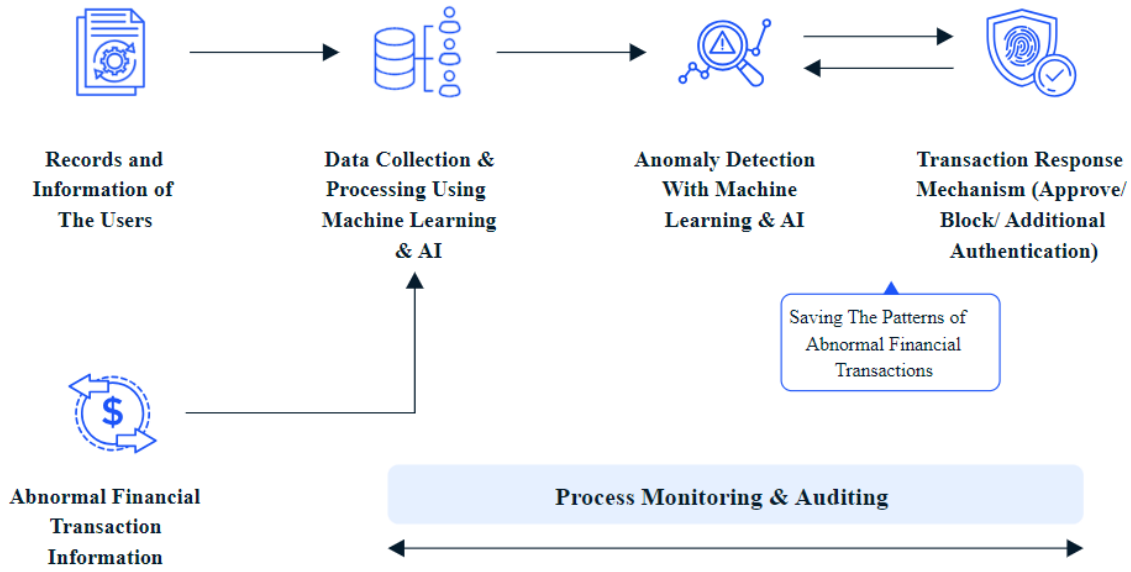
Rule-Based Fraud Detection



Machine Learning-Based Fraud Detection



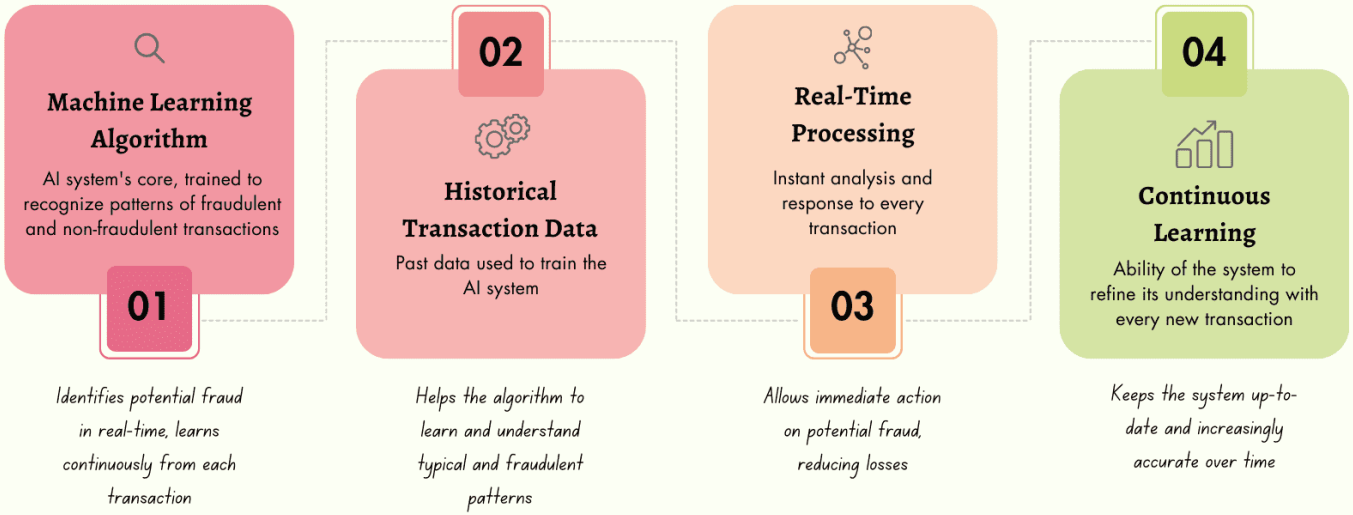
How does AI work in fraud detection?



LeewayHertz

capella

Components of an AI System for Fraud Detection



Detection

Reduce noise (false positives) and the risk that fraudulent transactions are missed (false negatives)

1 Algorithm Enhancements

Organizations implement powerful algorithms and sophisticated escalation approaches to verify and authenticate risky transactions, thereby minimizing the risk of false positives.

2 Usage Pattern Profiles

Usage pattern profiles enable the detection of previously unseen types of fraud attacks, enhancing the overall robustness of the detection system.

3 Closure Data Utilization

Faster identification of fraud attacks is achieved by using closure data in the detection queues, leading to reduced learning times and associated costs.

KPI	Why It Matters	Optimal Range
Fraud Detection Rate	Indicates the proportion of fraudulent activities correctly identified by the AI system	Aim for high (close to 100%)
False Positive Rate	Indicates the proportion of legitimate transactions mistakenly flagged as fraudulent by the AI system	Aim for low
Fraud Prevention Rate	Indicates the proportion of potential fraud successfully prevented due to timely detection	Aim for high
Time to Detect	Measures the average time taken by the AI system to detect potential fraud	Aim for low (real-time if possible)

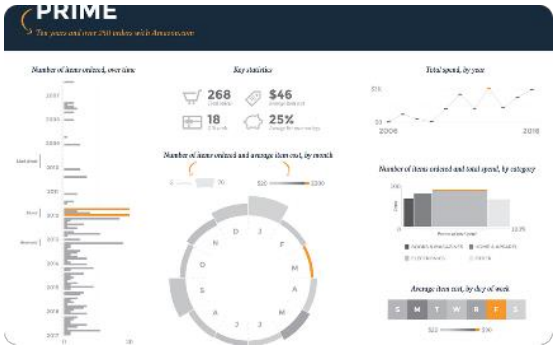
Utilizing Alternative Data Sources

Social Media Data

Social media platforms provide valuable information for fraud detection, including user behavior, communication patterns, and potential red flags.

Geospatial Data

Location-based data offers insights into the geographical patterns of transactions, enabling the identification of suspicious activities.



Purchasing History Data

Analyzing purchasing history allows for the detection of anomalies and unusual spending behaviors, indicating potential fraudulent activities.

Leveraging AI Techniques for Effective Fraud Detection

Logistic Regression

Provides a categorical output – either 'fraud' or 'non-fraud' – but false positives require costly manual investigations.

Decision Tree (DT) / Random Forest

A benefit of this method is that the generated DT can be viewed as rules in a similar form to that in expert systems.

Natural Language Processing (NLP)

By extracting signals from chat, voice, and IVR interactions, NLP enables these companies to identify and prevent fraudulent activities more effectively.

Neural Networks

Flexible computing systems applied to complex pattern recognition and prediction problems, clustering and forecasting behaviors

How about Gen AI?