The University of Faisalabad

MACHINE LEARNING

Submitted by: Ali Hassan

Registration: 2023-BSAI-038

Submitted to: SIR SAEED

LAB MANUAL

Crop yield prediction with linear regression

Stock market classification using machine learning

Crop yield prediction with linear regression

Introduction

This project aims to predict crop yield based on various environmental and agricultural factors using regression techniques. The model will help farmers and policymakers forecast crop production, optimize resource allocation, and enhance agricultural productivity.

Objective of the Project

The goal of this project is to develop a regression model that accurately predicts crop yield by analyzing features such as rainfall, temperature, soil quality, fertilizer usage, and other related variables.

Data Description

- Source: The dataset was collected from [source, e.g., Kaggle, government agricultural data, etc.].
- Shape: The dataset contains N rows (samples) and M columns (features + target variable).
- Features: The dataset includes environmental and agronomic features such as rainfall, temperature, humidity, soil pH, fertilizer amount, and seed variety.
- Target Variable: Crop yield (e.g., kg per hectare).
- Missing Values: Missing data were handled through imputation and removal techniques as described in preprocessing steps.

Libraries Installation

List and explain the libraries used for data handling, visualization, preprocessing, and modeling.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler, LabelEncoder, MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error, r2_score
```

```
import warnings
warnings.filterwarnings('ignore')
```

```
print("Libraries imported successfully.")
```

```
Libraries imported successfully.
```

Reading Data

Describe the process of loading the dataset into the programming environment and initial data inspection.

```
df = pd.read_csv('crop_recommendation.csv')
print("\nDataset Loaded Successfully. Displaying first 5 rows:")
display(df.head())
```

```
Dataset Loaded Successfully. Displaying first 5 rows:
```

	N	P	K	temperature	humidity	ph	rainfall	label
0	90	42	43	20.879744	82.002744	6.502985	202.935536	rice
1	85	58	41	21.770462	80.319644	7.038096	226.655537	rice
2	60	55	44	23.004459	82.320763	7.840207	263.964248	rice
3	74	35	40	26.491096	80.158363	6.980401	242.864034	rice
4	78	42	42	20.130175	81.604873	7.628473	262.717340	rice

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2200 entries, 0 to 2199
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   N            2200 non-null   int64  
 1   P            2200 non-null   int64  
 2   K            2200 non-null   int64  
 3   temperature  2200 non-null   float64
 4   humidity     2200 non-null   float64
 5   ph           2200 non-null   float64
 6   rainfall     2200 non-null   float64
 7   label        2200 non-null   object 
dtypes: float64(4), int64(3), object(1)
memory usage: 137.6+ KB
None
```

```
print(df.describe())
```

```
                 N            P            K  temperature     humidity  \
count  2200.000000  2200.000000  2200.000000  2200.000000  2200.000000   
mean     50.551818    53.362727    48.149091    25.616244    71.481779   
std      36.917334    32.985883    50.647931     5.063749    22.263812   
min       0.000000     5.000000     5.000000     8.825675    14.258040   
25%      21.000000    28.000000    20.000000    22.769375    60.261953   
50%      37.000000    51.000000    32.000000    25.598693    80.473146   
75%      84.250000    68.000000    49.000000    28.561654    89.948771   
max     140.000000   145.000000   205.000000    43.675493    99.981876   

                ph     rainfall  
count  2200.000000  2200.000000  
mean      6.469480   103.463655  
std       0.773938    54.958389  
min       3.504752    20.211267  
25%       5.971693    64.551686  
50%       6.425045    94.867624  
75%       6.923643   124.267508  
max       9.935091   298.560117  
```

```
print(f"Columns: {df.columns.tolist()}")
```

```
Columns: ['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall', 'label']
```

```
df.tail()
```

	N	P	K	temperature	humidity	ph	rainfall	label
2195	107	34	32	26.774637	66.413269	6.780064	177.774507	coffee
2196	99	15	27	27.417112	56.636362	6.086922	127.924610	coffee
2197	118	33	30	24.131797	67.225123	6.362608	173.322839	coffee
2198	117	32	34	26.272418	52.127394	6.758793	127.175293	coffee
2199	104	18	30	23.603016	60.396475	6.779833	140.937041	coffee

```
df.shape
```

```
(2200, 8)
```

```
df.sample()
```

	N	P	K	temperature	humidity	ph	rainfall	label
737	57	60	17	26.237731	67.885214	7.504608	73.58664	blackgram

Initial Preprocessing

- Removing duplicates
- Dropping irrelevant columns

```
duplicates = df.duplicated().sum()
print(f"\nNumber of duplicate rows: {duplicates}")

if duplicates > 0:
    df = df.drop_duplicates()
    print("Duplicates removed.")
else:
    print("No duplicates found.")
```

```
Number of duplicate rows: 0
No duplicates found.
```

```
df['label'] = df['label'].astype(str)
```

Handling Missing Values

- Detecting missing data

- Techniques: mean/mode imputation, deletion

```
missing_values = df.isnull().sum()
print("\nMissing values in each column:")
print(missing_values)
```

```
Missing values in each column:
N              0
P              0
K              0
temperature    0
humidity       0
ph             0
rainfall       0
label          0
dtype: int64
```

```
df_clean = df.dropna()
```

```
# Since missing values are critical, drop rows with missing values if any
if missing_values.sum() > 0:
    df = df.dropna()
    print("Rows with missing values dropped.")
else:
    print("No missing values found.")

print(f"\nShape after removing missing values: {df.shape}")
```

```
No missing values found.

Shape after removing missing values: (2200, 8)
```

Data Types & Conversion

- Ensuring correct data types for analysis and modeling

```
print("\nData types before conversion:")
print(df.dtypes)
```

```
Data types before conversion:
N                int64
P                int64
K                int64
temperature    float64
humidity       float64
ph             float64
rainfall       float64
label           object
dtype: object
```

```
df['label'] = df['label'].astype(str)
```

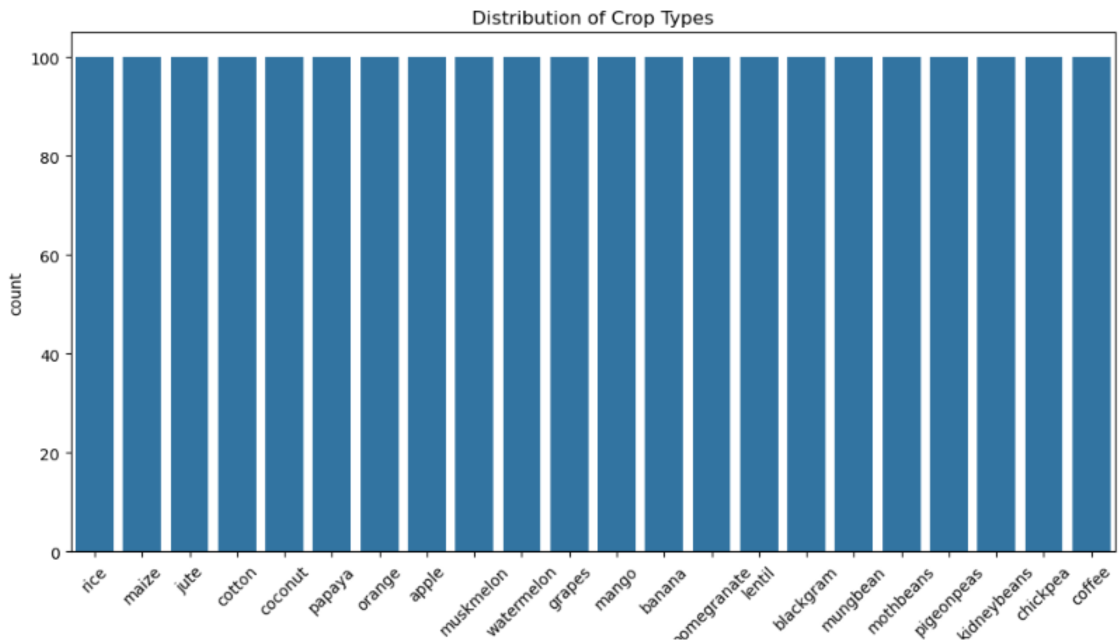
```
print("\nData types after conversion:")
print(df.dtypes)
```

```
Data types after conversion:
N                int64
P                int64
K                int64
temperature    float64
humidity       float64
ph             float64
rainfall       float64
label           object
dtype: object
```

Exploratory Data Analysis (EDA)

- Summary statistics
- Visualizations: histograms, boxplots, correlation matrix

```
plt.figure(figsize=(12,6))
sns.countplot(x='label', data=df, order=df['label'].value_counts().index)
plt.title('Distribution of Crop Types')
plt.xticks(rotation=45)
plt.show()
```

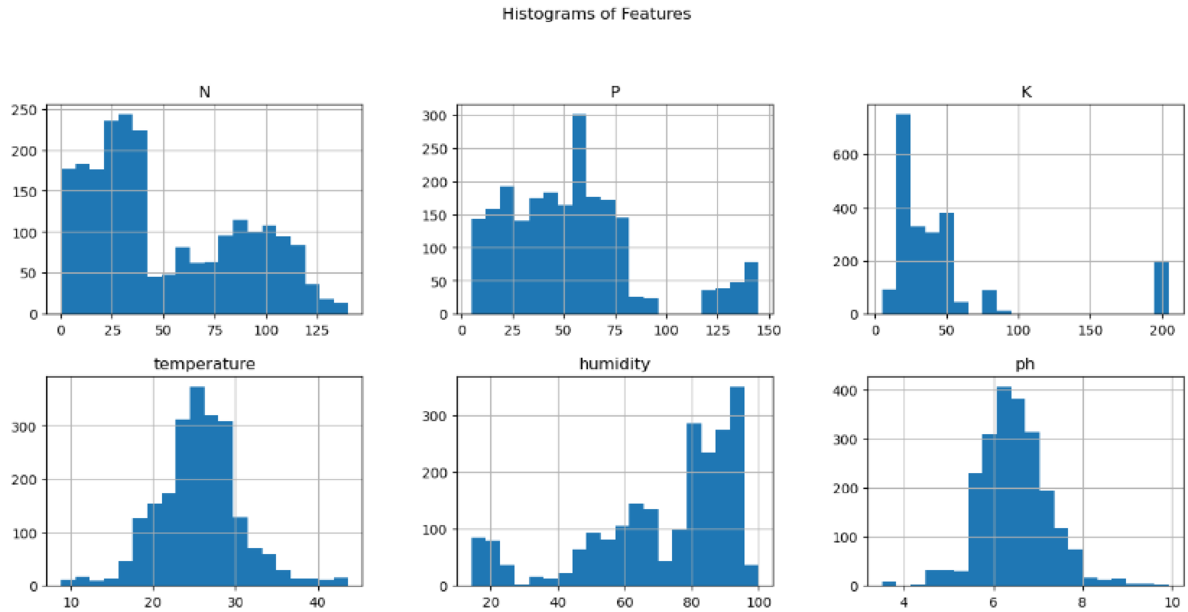



```
print("\nSummary Statistics of Numeric Features:")
display(df.describe())
```

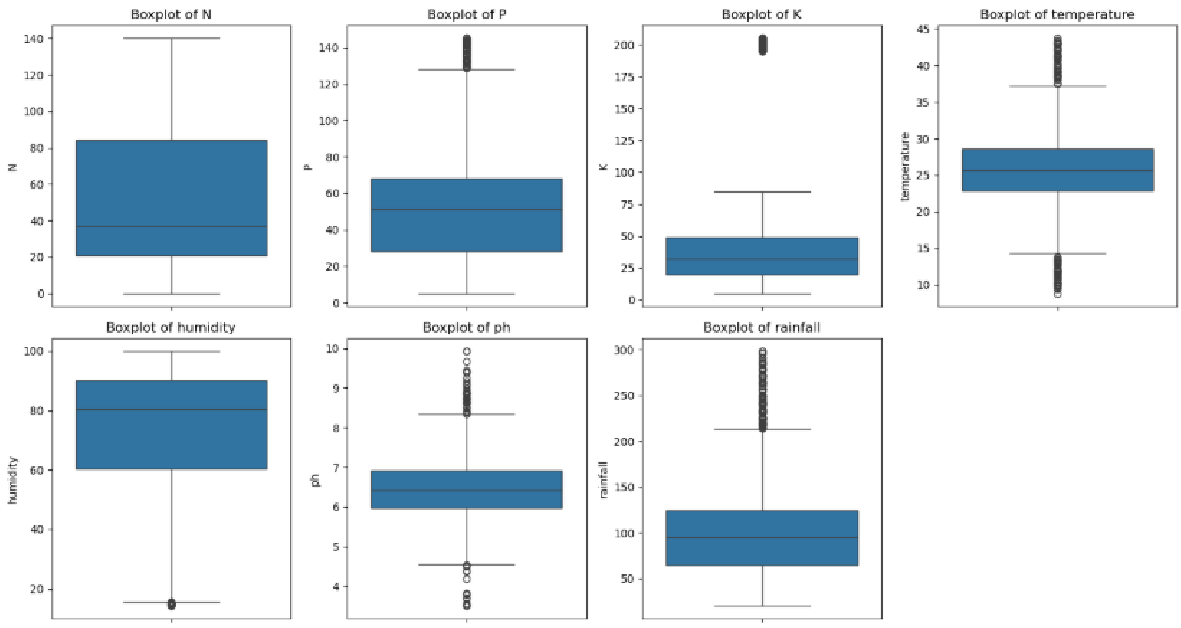
```
Summary Statistics of Numeric Features:
```

	N	P	K	temperature	humidity	ph	rainfall
count	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000	2200.000000
mean	50.551818	53.362727	48.149091	25.616244	71.481779	6.469480	103.463655
std	36.917334	32.985883	50.647931	5.063749	22.263812	0.773938	54.958389
min	0.000000	5.000000	5.000000	8.825675	14.258040	3.504752	20.211267
25%	21.000000	28.000000	20.000000	22.769375	60.261953	5.971693	64.551686
50%	37.000000	51.000000	32.000000	25.598693	80.473146	6.425045	94.867624
75%	84.250000	68.000000	49.000000	28.561654	89.948771	6.923643	124.267508
max	140.000000	145.000000	205.000000	43.675493	99.981876	9.935091	298.560117

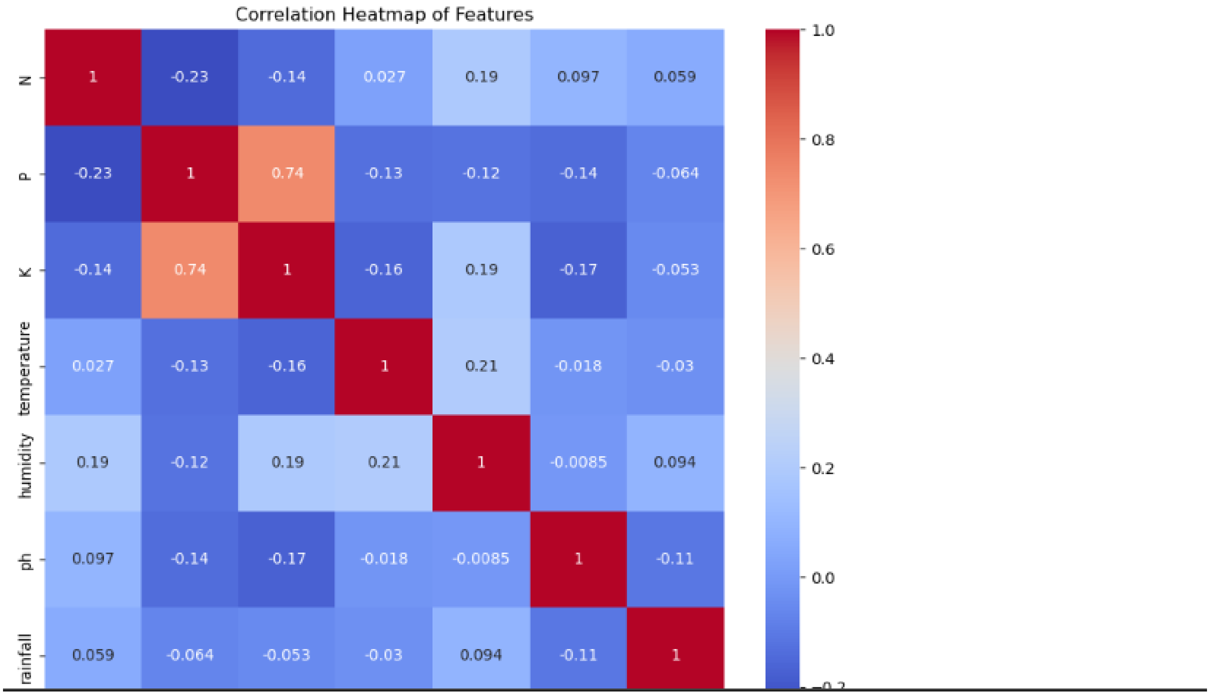
```
df.hist(column=['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall'], figsize=(15,10), bins=20)
plt.suptitle('Histograms of Features')
plt.show()
```




```
plt.figure(figsize=(15,8))
for i, col in enumerate(['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall']):
    plt.subplot(2,4,i+1)
    sns.boxplot(y=df[col])
    plt.title(f'Boxplot of {col}')
plt.tight_layout()
plt.show()
```



```
plt.figure(figsize=(10,8))
sns.heatmap(df[['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall']].corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap of Features')
plt.show()
```



Outlier Removal Using IQR Method

- Detecting outliers with Interquartile Range (IQR)
- Removing or capping outliers to improve model robustness

```
def remove_outliers(df, feature):
    Q1 = df[feature].quantile(0.25)
    Q3 = df[feature].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5*IQR
    upper_bound = Q3 + 1.5*IQR
    before = df.shape[0]
    df_filtered = df[(df[feature] >= lower_bound) & (df[feature] <= upper_bound)]
    after = df_filtered.shape[0]
    print(f'{feature}: Removed {before - after} outliers')
    return df_filtered
```

```
for feature in ['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall']:
    df = remove_outliers(df, feature)
```

```
N: Removed 0 outliers
P: Removed 138 outliers
K: Removed 62 outliers
temperature: Removed 58 outliers
humidity: Removed 0 outliers
ph: Removed 58 outliers
rainfall: Removed 38 outliers
```

```
print(f"\nShape after outlier removal: {df.shape}")
```

```
Shape after outlier removal: (1846, 8)
```

Feature Scaling

- Standardization or normalization of features

```
features = ['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall']
X = df[features]
```

```
minmax_scaler = MinMaxScaler()
X_norm = minmax_scaler.fit_transform(X)
X_norm_df = pd.DataFrame(X_norm, columns=features)
print("\nFirst 5 rows after MinMax Normalization:")
display(X_norm_df.head())
```

```
First 5 rows after MinMax Normalization:
```

	N	P	K	temperature	humidity	ph	rainfall
0	0.642857	0.411111	0.4750	0.259066	0.790267	0.500431	0.799151
1	0.607143	0.588889	0.4500	0.300649	0.770633	0.641414	0.902891
2	0.528571	0.333333	0.4375	0.521029	0.768751	0.626213	0.973780
3	0.671429	0.533333	0.4375	0.230962	0.800665	0.293780	0.969888
4	0.635714	0.544444	0.4125	0.428817	0.808144	0.548477	0.919470

```
scaler = StandardScaler()
X_std = scaler.fit_transform(X)
X_std_df = pd.DataFrame(X_std, columns=features)
print("\nFirst 5 rows after Standardization:")
display(X_std_df.head())
```

```
First 5 rows after Standardization:
```

	N	P	K	temperature	humidity	ph	rainfall
0	0.943866	-0.140507	0.617341	-1.163084	0.540214	0.014157	1.935598
1	0.812004	0.570599	0.496932	-0.948659	0.465882	0.800959	2.388110
2	0.521906	-0.451616	0.436727	0.187749	0.458760	0.716127	2.697324
3	1.049356	0.348379	0.436727	-1.308004	0.579578	-1.139126	2.680348
4	0.917494	0.392823	0.316318	-0.287749	0.607893	0.282292	2.460426

Label Encoding for Target Variable

- Encoding target if categorical (specific use case)

```
le = LabelEncoder()
df['label_encoded'] = le.fit_transform(df['label'])
```

```
print("\nUnique crops and their encoded labels:")
for crop, code in zip(le.classes_, range(len(le.classes_))):
    print(f"{crop} -> {code}")
```

```
Unique crops and their encoded labels:
banana -> 0
blackgram -> 1
chickpea -> 2
coconut -> 3
coffee -> 4
cotton -> 5
jute -> 6
kidneybeans -> 7
lentil -> 8
maize -> 9
mango -> 10
mothbeans -> 11
mungbean -> 12
muskmelon -> 13
orange -> 14
papaya -> 15
pigeonpeas -> 16
pomegranate -> 17
rice -> 18
watermelon -> 19
```

```
y = df['label_encoded']
```

Train-Test Split & PCA

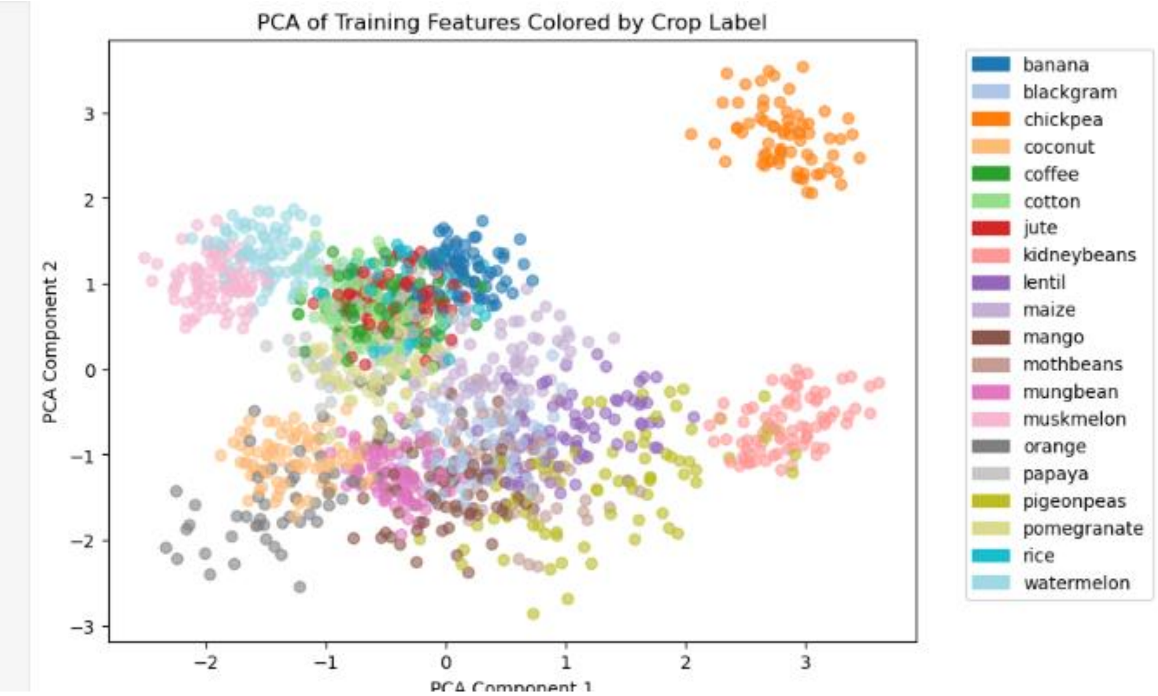
- Splitting data into training/testing sets
- Applying Principal Component Analysis for dimensionality reduction

```
X_train, X_test, y_train, y_test = train_test_split(X_std, y, test_size=0.2, random_state=42, stratify=y)
print(f"\nTrain set size: {X_train.shape[0]} samples")
print(f"Test set size: {X_test.shape[0]} samples")
```

```
Train set size: 1476 samples
Test set size: 370 samples
```

```
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

```
import matplotlib.patches as mpatches
import matplotlib.cm as cm
import numpy as np

plt.figure(figsize=(8,6))
scatter = plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train, cmap='tab20', alpha=0.6)
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.title('PCA of Training Features Colored by Crop Label')

# Create manual legend
unique_labels = np.unique(y_train)
cmap = cm.get_cmap('tab20', len(unique_labels))  # discrete colormap with required colors

patches = []
for i, label in enumerate(unique_labels):
    patches.append(mpatches.Patch(color=cmap(i), label=le.inverse_transform([label])[0]))

plt.legend(handles=patches, bbox_to_anchor=(1.05,1), loc='upper left')
plt.show()
```




```
print(f"\nExplained variance by 2 PCA components: {pca.explained_variance_ratio_.sum():.2%}")
```

```
Explained variance by 2 PCA components: 43.95%
```

Model Training - Linear Regression

- Fitting the linear regression model on training data

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```
lr_model = LinearRegression()

lr_model.fit(X_train, y_train)
```

```
LinearRegression
LinearRegression()
```

```
print(f"Intercept (bias term): {lr_model.intercept_}")
coefficients = pd.DataFrame(lr_model.coef_, index=features, columns=['Coefficient'])
print("Coefficients for each feature:")
print(coefficients)
```

```
Intercept (bias term): 9.15858756827002
Coefficients for each feature:
             Coefficient
N              -1.274039
P              -1.854094
K               0.177447
temperature    -0.534233
humidity        1.447463
ph             -0.590256
rainfall       -0.584797
```

Prediction

- Using the model to predict crop yield on test data

```
y_train_pred = lr_model.predict(X_train)
y_test_pred = lr_model.predict(X_test)
```

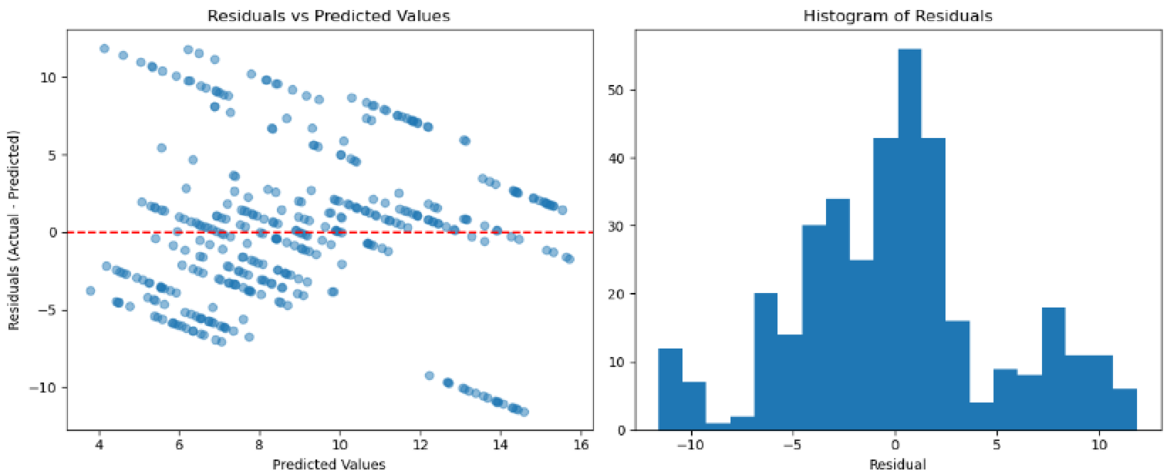
```
mse_train = mean_squared_error(y_train, y_train_pred)
rmse_train = np.sqrt(mse_train)
mae_train = mean_absolute_error(y_train, y_train_pred)
r2_train = r2_score(y_train, y_train_pred)

print("\nTraining Set Performance:")
print(f"Mean Squared Error (MSE): {mse_train:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse_train:.4f}")
print(f"Mean Absolute Error (MAE): {mae_train:.4f}")
print(f"R-squared (R2): {r2_train:.4f}")
```

```
Training Set Performance:
Mean Squared Error (MSE): 24.8910
Root Mean Squared Error (RMSE): 4.9891
Mean Absolute Error (MAE): 3.7450
R-squared (R2): 0.2475
```

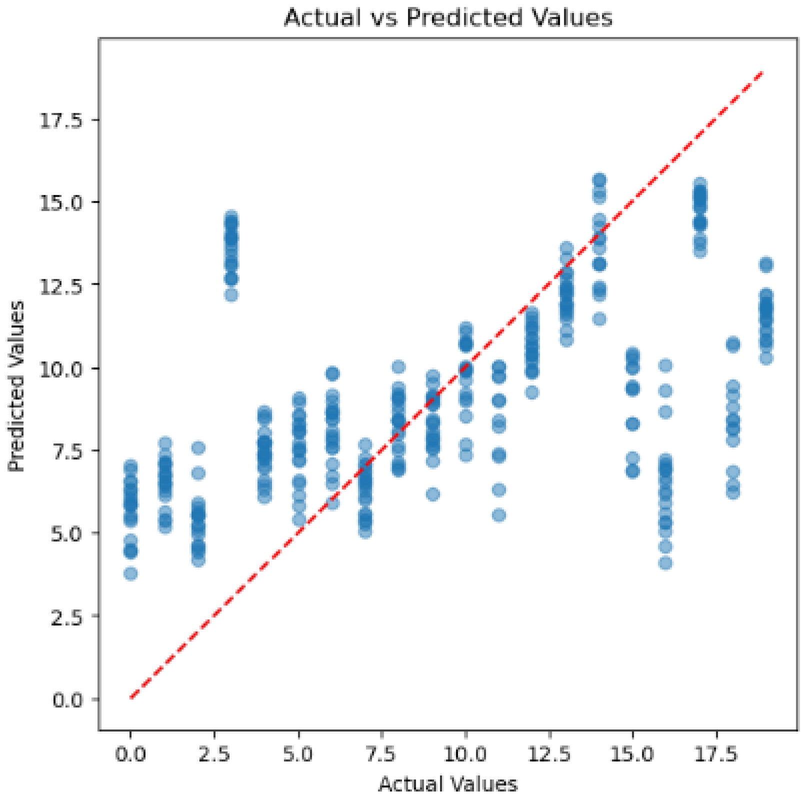
```
mse_test = mean_squared_error(y_test, y_test_pred)
rmse_test = np.sqrt(mse_test)
mae_test = mean_absolute_error(y_test, y_test_pred)
r2_test = r2_score(y_test, y_test_pred)

print("\nTest Set Performance:")
print(f"Mean Squared Error (MSE): {mse_test:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse_test:.4f}")
print(f"Mean Absolute Error (MAE): {mae_test:.4f}")
print(f"R-squared (R2): {r2_test:.4f}")
```

```
Test Set Performance:
Mean Squared Error (MSE): 24.8483
Root Mean Squared Error (RMSE): 4.9848
Mean Absolute Error (MAE): 3.7649
R-squared (R2): 0.2519
```

```
residuals = y_test - y_test_pred

plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
plt.scatter(y_test_pred, residuals, alpha=0.5)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals (Actual - Predicted)')
plt.title('Residuals vs Predicted Values')

plt.subplot(1,2,2)
plt.hist(residuals, bins=20)
plt.xlabel('Residual')
plt.title('Histogram of Residuals')

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

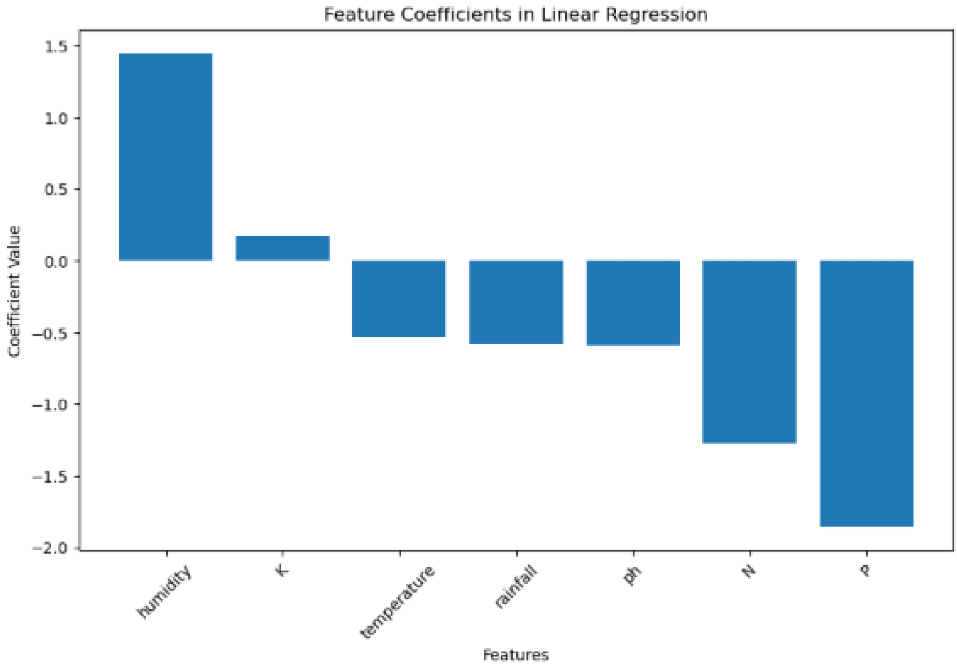


```
plt.figure(figsize=(6,6))
plt.scatter(y_test, y_test_pred, alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values')
plt.show()
```



```
explained_variance = r2_test
print(f"\nExplained Variance (R2) on Test Set: {explained_variance:.4f}")

coefficients_sorted = coefficients.sort_values(by='Coefficient', ascending=False)
plt.figure(figsize=(10,6))
plt.bar(coefficients_sorted.index, coefficients_sorted['Coefficient'])
plt.xticks(rotation=45)
plt.xlabel('Features')
plt.ylabel('Coefficient Value')
plt.title('Feature Coefficients in Linear Regression')
plt.show()
```

```
Explained Variance (R2) on Test Set: 0.2519
```



Summary of Project

This project worked on predicting how much crop will be produced by using data about weather and farming. We looked at important factors like rainfall, temperature, soil quality, and fertilizer use. The data was cleaned and prepared carefully to remove errors and missing information. Then, we used a method called linear regression to build a model that can estimate crop yield. The model gave fairly good predictions, which can help farmers and planners make better decisions. In the future, the model can be improved by using more data and more advanced techniques.

LAB MANUAL

Stock market classification using machine learning

Introduction

This project aims to classify stock market data using machine learning classification techniques to assist investors and analysts in predicting market trends and making informed decisions.

Objective of the Project

The goal of this project is to build a classification model that can accurately categorize stock market states or stock types by analyzing market indicators, historical prices, and volume data.

Data Description

- Source: The dataset was collected from [source, e.g., financial data APIs, Yahoo Finance, Kaggle].
- Shape: The dataset contains N rows (samples) and M columns (features + target variable).
- Features: The dataset includes stock market indicators such as open, close, high, low prices, volume, and technical indicators like moving averages and RSI.
- Target Variable: Stock market classification labels (e.g., buy/sell/hold or stock categories).
- Missing Values: Missing values were managed using imputation and removal methods during data preprocessing.

Libraries Installation & Imports

- pandas, numpy for data handling
- matplotlib, seaborn for plots
- scikit-learn, imblearn, keras/tensorflow for modeling

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
```

```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, roc_auc_score, roc_curve

from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

```
import warnings
warnings.filterwarnings('ignore')

print("All required libraries imported successfully.")
```

```
All required libraries imported successfully.
```

Reading Data

- Loading and inspecting stock data

```
df = pd.read_csv('2018_Financial_Data.csv')  # Change to your dataset path
print("\nDataset Loaded Successfully. Displaying first 5 rows:")
display(df.head())
```

```
Dataset Loaded Successfully. Displaying first 5 rows:
```

	Unnamed: 0	Revenue	Revenue Growth	Cost of Revenue	Gross Profit	R&D Expenses	SG&A Expense	Operating Expenses	Operating Income	Interest Expense	...	Receivables growth	Inventory Growth
0	CMCSA	9.450700e+10	0.1115	0.000000e+00	9.450700e+10	0.000000e+00	6.482200e+10	7.549800e+10	1.900900e+10	3.542000e+09	...	0.2570	0.000
1	KMI	1.414400e+10	0.0320	7.288000e+09	6.856000e+09	0.000000e+00	6.010000e+08	3.062000e+09	3.794000e+09	1.917000e+09	...	0.0345	-0.092
2	INTC	7.084800e+10	0.1289	2.711100e+10	4.373700e+10	1.354300e+10	6.750000e+09	2.042100e+10	2.331600e+10	-1.260000e+08	...	0.1989	0.038
3	MU	3.039100e+10	0.4955	1.250000e+10	1.789100e+10	2.141000e+09	8.130000e+08	2.897000e+09	1.499400e+10	3.420000e+08	...	0.4573	0.151
4	GE	1.216150e+11	0.0285	9.546100e+10	2.615400e+10	0.000000e+00	1.811100e+10	4.071100e+10	-1.455700e+10	5.059000e+09	...	-0.2781	-0.289

5 rows × 225 columns

```python
print("\nBasic info about data types and null counts:")
print(df.info())
```

```
Basic info about data types and null counts:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4392 entries, 0 to 4391
Columns: 225 entries, Unnamed: 0 to Class
dtypes: float64(222), int64(1), object(2)
memory usage: 7.5+ MB
None
```

```python
print(df.describe())
```

```
             Revenue  Revenue Growth  Cost of Revenue  Gross Profit  \
count  4.346000e+03     4253.000000     4.207000e+03  4.328000e+03
mean   5.119287e+09        3.455278     3.144946e+09  2.043954e+09
std    2.049504e+10      195.504906     1.508813e+10  7.682369e+09
min   -6.894100e+07       -3.461500    -2.669055e+09 -1.818220e+09
25%    6.501425e+07        0.000000     3.415500e+06  3.618903e+07
50%    4.982640e+08        0.074900     1.741180e+08  2.219470e+08
75%    2.457878e+09        0.188500     1.297814e+09  9.767015e+08
max    5.003430e+11    12739.000000     3.733960e+11  1.269470e+11

       R&D Expenses  SG&A Expense  Operating Expenses  Operating Income  \
count  4.155000e+03  4.226000e+03        4.208000e+03      4.357000e+03
mean   1.180176e+08  9.005022e+08        1.435546e+09      6.541207e+08
std    9.330891e+08  3.661116e+09        5.529831e+09      2.969341e+09
min   -1.042000e+08 -1.401594e+08       -4.280000e+09     -1.455700e+10
25%    0.000000e+00  2.056226e+07        4.223644e+07     -5.510000e+06
50%    0.000000e+00  9.390450e+07        1.806253e+08      4.203800e+07
75%    1.450150e+07  4.117162e+08        6.796040e+08      2.862690e+08
max    2.883700e+10  1.065100e+11        1.065100e+11      7.089800e+10

       Interest Expense  Earnings before Tax  ...  \
count      4.208000e+03         4.321000e+03  ...
mean       1.001350e+08         5.584432e+08  ...
std        3.780021e+08         2.639327e+09  ...
min       -1.408252e+09        -2.177200e+10  ...
25%        0.000000e+00        -1.000800e+07  ...
50%        5.693500e+06         2.730700e+07  ...
75%        5.817075e+07         2.238810e+08  ...
max        9.168000e+09         7.290300e+10  ...
```

```
       3Y Dividend per Share Growth (per Share)  Receivables growth  \
count                               4067.000000         4268.000000
mean                                   0.006081           36.768524
std                                    0.239653         2347.079237
min                                   -1.000000           -1.000000
25%                                    0.000000           -0.048075
50%                                    0.000000            0.010200
75%                                    0.042050            0.185900
max                                    4.079100       153332.333300

       Inventory Growth  Asset Growth  Book Value per Share Growth  \
count       4160.000000   4178.000000                  4121.000000
mean           0.183066      1.389013                     0.262530
std            4.688013     35.123904                     5.612666
min           -1.000000     -0.999100                   -32.258100
25%            0.000000     -0.036700                    -0.108600
50%            0.000000      0.034750                     0.026100
75%            0.080050      0.160575                     0.138400
max          293.473000   1184.993800                   313.395800

        Debt Growth  R&D Expense Growth  SG&A Expenses Growth  \
count   4128.000000         4133.000000           4144.000000
mean       9.928446            0.091891              0.153610
std      363.717734            0.823281              0.839647
min       -1.000000           -1.000000             -1.000000
25%       -0.082850            0.000000             -0.004650
50%        0.000000            0.000000              0.065700
75%        0.115425            0.009700              0.167625
max    17646.823500           36.898100             43.718800

       2019 PRICE VAR [%]        Class
count         4392.000000  4392.000000
mean            20.803948     0.693534
std             82.622147     0.461078
min            -99.864779     0.000000
25%             -7.477173     0.000000
50%             17.639393     1.000000
75%             39.625879     1.000000
max           3756.716345     1.000000

[8 rows x 223 columns]
```

```python
print(f"Columns: {df.columns.tolist()}")
```

```
Columns: ['Unnamed: 0', 'Revenue', 'Revenue Growth', 'Cost of Revenue', 'Gross Profit', 'R&D Expenses', 'SG&A Expense', 'Operating Expenses', 'Operating
Income', 'Interest Expense', 'Earnings before Tax', 'Income Tax Expense', 'Net Income - Non-Controlling int', 'Net Income - Discontinued ops', 'Net Inco
me', 'Preferred Dividends', 'Net Income Com', 'EPS', 'EPS Diluted', 'Weighted Average Shs Out', 'Weighted Average Shs Out (Dil)', 'Dividend per Share',
'Gross Margin', 'EBITDA Margin', 'EBIT Margin', 'Profit Margin', 'Free Cash Flow margin', 'EBITDA', 'EBIT', 'Consolidated Income', 'Earnings Before Tax
Margin', 'Net Profit Margin', 'Cash and cash equivalents', 'Short-term investments', 'Cash and short-term investments', 'Receivables', 'Inventories', 'T
otal current assets', 'Property, Plant & Equipment Net', 'Goodwill and Intangible Assets', 'Long-term investments', 'Tax assets', 'Total non-current ass
ets', 'Total assets', 'Payables', 'Short-term debt', 'Total current liabilities', 'Long-term debt', 'Total debt', 'Deferred revenue', 'Tax Liabilities',
'Deposit Liabilities', 'Total non-current liabilities', 'Total liabilities', 'Other comprehensive income', 'Retained earnings (deficit)', 'Total shareho
lders equity', 'Investments', 'Net Debt', 'Other Assets', 'Other Liabilities', 'Depreciation & Amortization', 'Stock-based compensation', 'Operating Cas
h Flow', 'Capital Expenditure', 'Acquisitions and disposals', 'Investment purchases and sales', 'Investing Cash flow', 'Issuance (repayment) of debt',
'Issuance (buybacks) of shares', 'Dividend payments', 'Financing Cash Flow', 'Effect of forex changes on cash', 'Net cash flow / Change in cash', 'Free
Cash Flow', 'Net Cash/Marketcap', 'priceBookValueRatio', 'priceToBookRatio', 'priceToSalesRatio', 'priceEarningsRatio', 'priceToFreeCashFlowsRatio', 'pr
iceToOperatingCashFlowsRatio', 'priceCashFlowRatio', 'priceEarningsToGrowthRatio', 'priceSalesRatio', 'dividendYield', 'enterpriseValueMultiple', 'price
FairValue', 'ebitperRevenue', 'ebtperEBIT', 'niperEBT', 'grossProfitMargin', 'operatingProfitMargin', 'pretaxProfitMargin', 'netProfitMargin', 'effectiv
eTaxRate', 'returnOnAssets', 'returnOnEquity', 'returnOnCapitalEmployed', 'nIperEBT', 'eBTperEBIT', 'eBITperRevenue', 'payablesTurnover', 'inventoryTurn
over', 'fixedAssetTurnover', 'assetTurnover', 'currentRatio', 'quickRatio', 'cashRatio', 'daysOfSalesOutstanding', 'daysOfInventoryOutstanding', 'operat
ingCycle', 'daysOfPayablesOutstanding', 'cashConversionCycle', 'debtRatio', 'debtEquityRatio', 'longtermDebtToCapitalization', 'totalDebtToCapitalizatio
n', 'interestCoverage', 'cashFlowToDebtRatio', 'companyEquityMultiplier', 'operatingCashFlowPerShare', 'freeCashFlowPerShare', 'cashPerShare', 'payoutRa
tio', 'operatingCashFlowSalesRatio', 'freeCashFlowOperatingCashFlowRatio', 'cashFlowCoverageRatios', 'shortTermCoverageRatios', 'capitalExpenditureCover
ageRatios', 'dividendpaidAndCapexCoverageRatios', 'dividendPayoutRatio', 'Revenue per Share', 'Net Income per Share', 'Operating Cash Flow per Share',
'Free Cash Flow per Share', 'Cash per Share', 'Book Value per Share', 'Tangible Book Value per Share', 'Shareholders Equity per Share', 'Interest Debt p
er Share', 'Market Cap', 'Enterprise Value', 'PE ratio', 'Price to Sales Ratio', 'POCF ratio', 'PFCF ratio', 'PB ratio', 'PTB ratio', 'EV to Sales', 'En
terprise Value over EBITDA', 'EV to Operating cash flow', 'EV to Free cash flow', 'Earnings Yield', 'Free Cash Flow Yield', 'Debt to Equity', 'Debt to A
ssets', 'Net Debt to EBITDA', 'Current ratio', 'Interest Coverage', 'Income Quality', 'Dividend Yield', 'Payout Ratio', 'SG&A to Revenue', 'R&D to Reven
ue', 'Intangibles to Total Assets', 'Capex to Operating Cash Flow', 'Capex to Revenue', 'Capex to Depreciation', 'Stock-based compensation to Revenue',
'Graham Number', 'ROIC', 'Return on Tangible Assets', 'Graham Net-Net', 'Working Capital', 'Tangible Asset Value', 'Net Current Asset Value', 'Invested
Capital', 'Average Receivables', 'Average Payables', 'Average Inventory', 'Days Sales Outstanding', 'Days Payables Outstanding', 'Days of Inventory on H
and', 'Receivables Turnover', 'Payables Turnover', 'Inventory Turnover', 'ROE', 'Capex per Share', 'Gross Profit Growth', 'EBIT Growth', 'Operating Inco
me Growth', 'Net Income Growth', 'EPS Growth', 'EPS Diluted Growth', 'Weighted Average Shares Growth', 'Weighted Average Shares Diluted Growth', 'Divide
nds per Share Growth', 'Operating Cash Flow growth', 'Free Cash Flow growth', '10Y Revenue Growth (per Share)', '5Y Revenue Growth (per Share)', '3Y Rev
enue Growth (per Share)', '10Y Operating CF Growth (per Share)', '5Y Operating CF Growth (per Share)', '3Y Operating CF Growth (per Share)', '10Y Net In
come Growth (per Share)', '5Y Net Income Growth (per Share)', '3Y Net Income Growth (per Share)', '10Y Shareholders Equity Growth (per Share)', '5Y Shar
eholders Equity Growth (per Share)', '3Y Shareholders Equity Growth (per Share)', '10Y Dividend per Share Growth (per Share)', '5Y Dividend per Share Gr
owth (per Share)', '3Y Dividend per Share Growth (per Share)', 'Receivables growth', 'Inventory Growth', 'Asset Growth', 'Book Value per Share Growth',
'Debt Growth', 'R&D Expense Growth', 'SG&A Expenses Growth', 'Sector', '2019 PRICE VAR [%]', 'Class']
```

```
df.tail()
```

	Unnamed: 0	Revenue	Revenue Growth	Cost of Revenue	Gross Profit	R&D Expenses	SG&A Expense	Operating Expenses	Operating Income	Interest Expense	...	Receivables growth	Inventory Growth	Asset Growth	Bo Val p Sha Grow
4387	YRIV	0.0	0.0000	0.0	0.0	0.0	3755251.0	3755251.0	-3755251.0	11058486.0	...	0.0000	0.0000	-0.0508	-0.14
4388	YTEN	556000.0	-0.4110	0.0	556000.0	4759000.0	5071000.0	9830000.0	-9274000.0	0.0	...	0.3445	0.0000	-0.2323	-0.86
4389	ZKIN	54884381.0	0.2210	36593792.0	18290589.0	1652633.0	7020320.0	8672953.0	9617636.0	1239170.0	...	0.1605	0.7706	0.2489	0.40
4390	ZOM	0.0	0.0000	0.0	0.0	10317153.0	4521349.0	16648628.0	-16648628.0	0.0	...	0.8980	0.0000	0.1568	-0.22
4391	ZYME	53019000.0	0.0243	0.0	53019000.0	56684000.0	29457000.0	86146000.0	-33127000.0	166000.0	...	-0.4185	0.0000	0.8519	0.13

5 rows × 225 columns

```
df.shape
```

```
(4392, 225)
```

```
df.sample()
```

	Unnamed: 0	Revenue	Revenue Growth	Cost of Revenue	Gross Profit	R&D Expenses	SG&A Expense	Operating Expenses	Operating Income	Interest Expense	...	Receivables growth	Inventory Growth	Asset Growth	Book Value per Share Growth	(
3276	WHG	122300000.0	-0.0858	0.0	122300000.0	0.0	71459000.0	89153000.0	33147000.0	0.0	...	-0.1492	0.0	0.0339	0.0036	

1 rows × 225 columns

Exploratory Data Analysis (EDA)

- Statistical summaries
- Visualizing class distribution and feature behavior

```
print("\nTarget variable ('Class') distribution:")
print(df['Class'].value_counts())

plt.figure(figsize=(6,4))
sns.countplot(x='Class', data=df)
plt.title('Class Distribution (0 = Not Buy, 1 = Buy)')
plt.show()
```

```
Target variable ('Class') distribution:
Class
1    3046
0    1346
Name: count, dtype: int64
```


```
missing_counts = df.isnull().sum()
missing_percents = 100 * missing_counts / len(df)
missing_df = pd.DataFrame({'Missing Count': missing_counts, 'Missing Percent': missing_percents})
missing_df = missing_df[missing_df['Missing Count'] > 0].sort_values(by='Missing Percent', ascending=False)
print("\nColumns with missing values sorted by percent:")
display(missing_df)
```

```
Columns with missing values sorted by percent:
```

	Missing Count	Missing Percent
operatingCycle	4386	99.863388
cashConversionCycle	4386	99.863388
shortTermCoverageRatios	1926	43.852459
10Y Shareholders Equity Growth (per Share)	1695	38.592896
dividendPayoutRatio	1658	37.750455
...	...	...
Operating Income	35	0.796903
Long-term debt	30	0.683060
Net cash flow / Change in cash	24	0.546448
Retained earnings (deficit)	21	0.478142
Financing Cash Flow	19	0.432605

221 rows × 2 columns

```
num_cols = df.select_dtypes(include=['float64', 'int64']).columns.tolist()
cat_cols = df.select_dtypes(include=['object']).columns.tolist()

print(f"\nTotal numerical columns: {len(num_cols)}")
print(f"Total categorical columns: {len(cat_cols)}")
```

```
Total numerical columns: 223
Total categorical columns: 2
```

```
print(f"Categorical columns: {cat_cols}")
```

```
Categorical columns: ['Unnamed: 0', 'Sector']
```

Data Cleaning

- Removing duplicates and irrelevant data
- Correcting inconsistent entries

```
num_duplicates = df.duplicated().sum()
print(f"\nNumber of duplicate rows: {num_duplicates}")
if num_duplicates > 0:
    df.drop_duplicates(inplace=True)
    print("Duplicates removed.")
else:
    print("No duplicates found.")
```

```
Number of duplicate rows: 0
No duplicates found.
```

```
print("\nMissing values count per column after duplicate removal:")
print(df.isnull().sum().sum())
```

```
Missing values count per column after duplicate removal:
97298
```

Handling Missing Values

- Detect and impute or remove missing values

```
from sklearn.impute import SimpleImputer

num_cols.remove('Class')  # Target should not be imputed

num_imputer = SimpleImputer(strategy='median')
cat_imputer = SimpleImputer(strategy='most_frequent')

df[num_cols] = num_imputer.fit_transform(df[num_cols])
if len(cat_cols) > 0:
    df[cat_cols] = cat_imputer.fit_transform(df[cat_cols])

print("\nAfter imputation, missing values count:")
print(df.isnull().sum().sum())  # Should be zero
```

```
After imputation, missing values count:
0
```

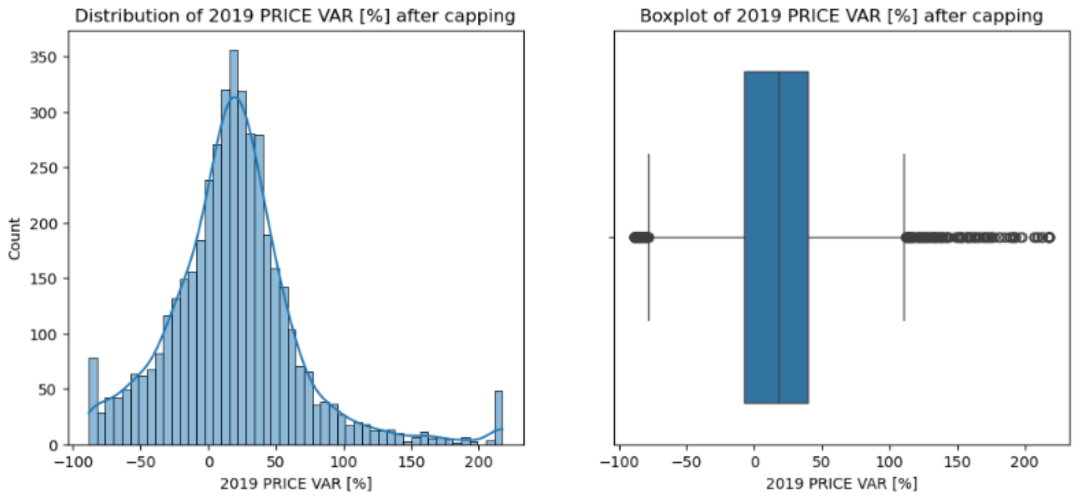
Outlier Detection and Treatment

- Identify outliers and treat them to avoid skewing models

```
for col in num_cols:
    lower = df[col].quantile(0.01)
    upper = df[col].quantile(0.99)
    before_outliers = df.shape[0]
    df[col] = np.where(df[col] < lower, lower, df[col])
    df[col] = np.where(df[col] > upper, upper, df[col])
print("\nOutliers capped at 1st and 99th percentiles for numerical features.")
```

```
Outliers capped at 1st and 99th percentiles for numerical features.
```

```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
sns.histplot(df[col], bins=50, kde=True)
plt.title(f'Distribution of {col} after capping')
plt.subplot(1,2,2)
sns.boxplot(x=df[col])
plt.title(f'Boxplot of {col} after capping')
plt.show()
```



Data Transformation: Scaling & Encoding

- Scale numerical features
- Encode categorical variables

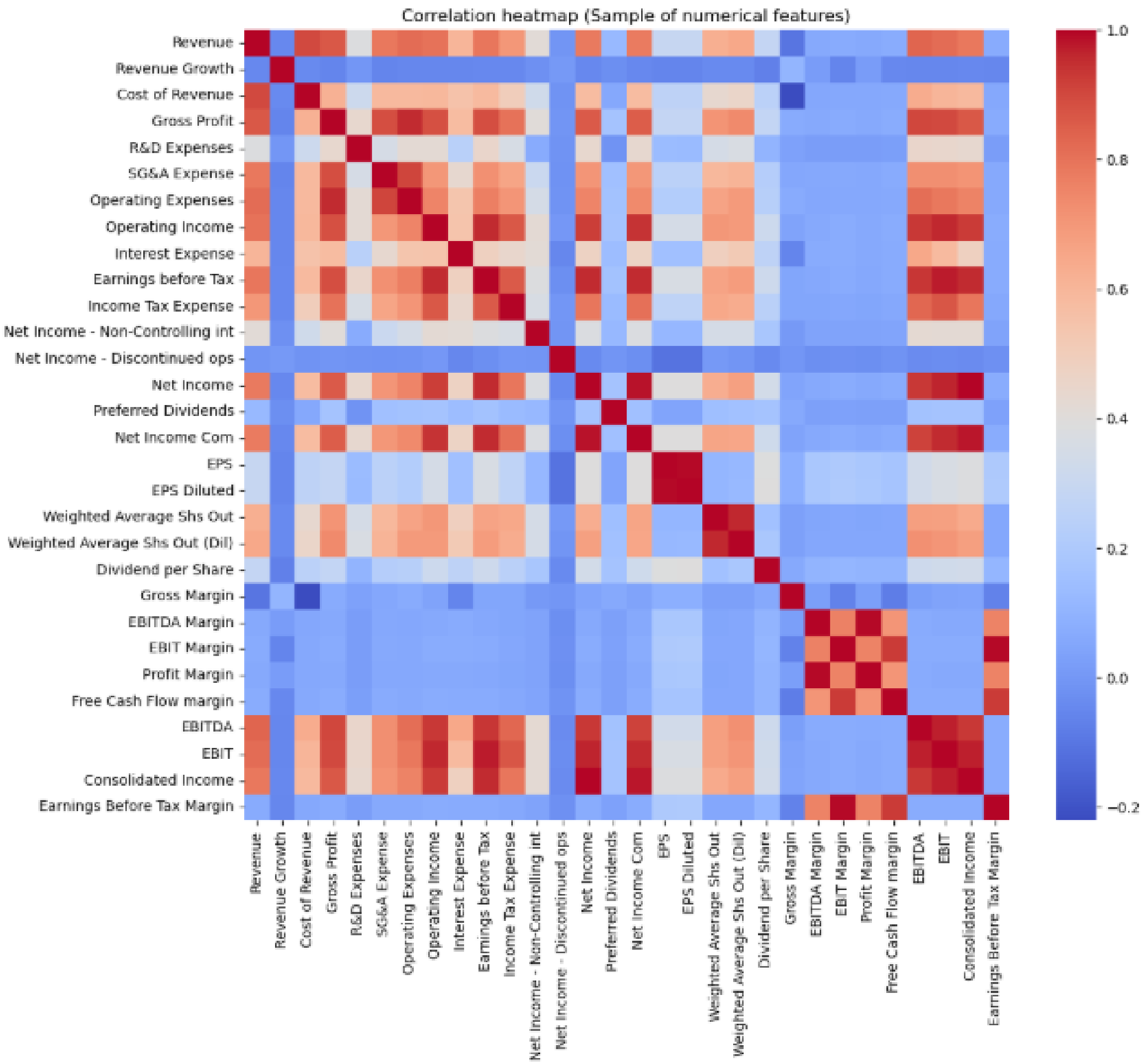
```
# Verify which categorical columns still exist
existing_cat_cols = [col for col in cat_cols if col in df.columns]

print(f"Categorical columns present for encoding: {existing_cat_cols}")

if existing_cat_cols:
    df = pd.get_dummies(df, columns=existing_cat_cols, drop_first=True)
    print(f"One-hot encoded columns: {existing_cat_cols}")
else:
    print("No categorical columns present for one-hot encoding.")
```

```
Categorical columns present for encoding: ['Unnamed: 0', 'Sector']
One-hot encoded columns: ['Unnamed: 0', 'Sector']
```

Split Data into Features and Target

- Separate independent variables and dependent target

```
X = df.drop(columns=['Class'])
y = df['Class']

print(f"\nFeature matrix shape: {X.shape}")
print(f"Target vector shape: {y.shape}")

sample_num_cols = num_cols[:30]
plt.figure(figsize=(12,10))
sns.heatmap(df[sample_num_cols].corr(), cmap='coolwarm', annot=False)
plt.title('Correlation heatmap (Sample of numerical features)')
plt.show()
```



Train-Test Split

- Partition dataset into training and testing sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

```
print(f"\nTraining set size: {X_train.shape[0]} samples")
print(f"Test set size: {X_test.shape[0]} samples")
```

```
Training set size: 3513 samples
Test set size: 879 samples
```

```
print("\nClass distribution in training data:")
print(y_train.value_counts(normalize=True))
```

```
Class distribution in training data:
Class
1    0.693424
0    0.306576
Name: proportion, dtype: float64
```

Handling Imbalanced Data with SMOTE

- Apply Synthetic Minority Over-sampling Technique to balance classes

```
smote = SMOTE(random_state=42)
X_train_sm, y_train_sm = smote.fit_resample(X_train, y_train)
```

```
print("\nAfter SMOTE oversampling:")
print(pd.Series(y_train_sm).value_counts(normalize=True))
```

```
After SMOTE oversampling:
Class
1    0.5
0    0.5
Name: proportion, dtype: float64
```

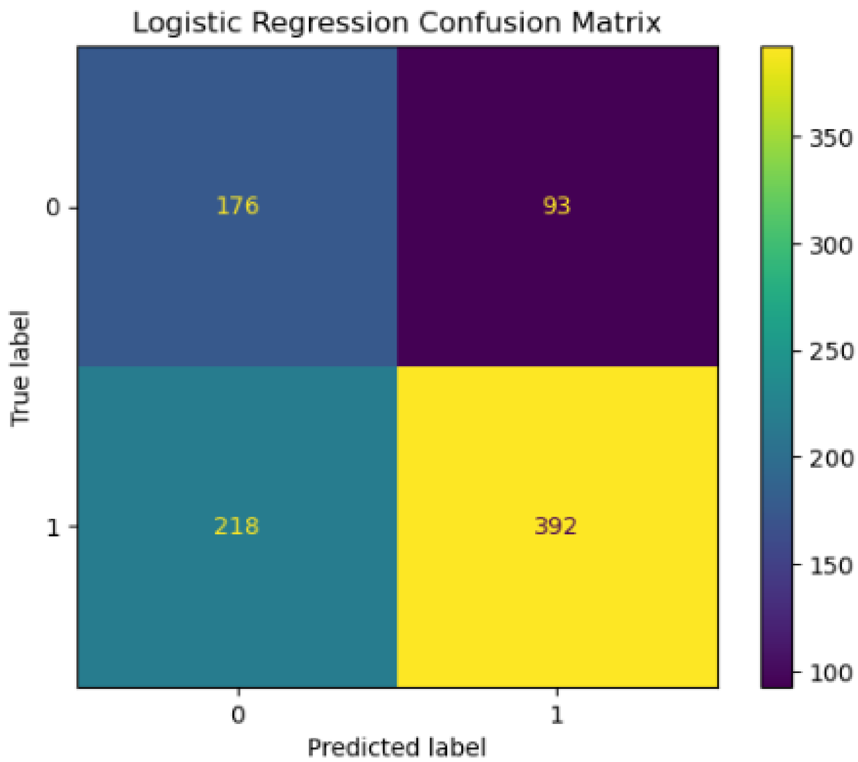
Building and Evaluating Models

- Train multiple classification algorithms
- Evaluate with accuracy, precision, recall, F1-score

```
from sklearn.metrics import ConfusionMatrixDisplay

def evaluate_model(model, X_test, y_test, model_name):
    y_pred = model.predict(X_test)
    print(f"\n--- {model_name} Performance ---")
    print(classification_report(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
    ConfusionMatrixDisplay(cm).plot()
    plt.title(f'{model_name} Confusion Matrix')
    plt.show()
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
```

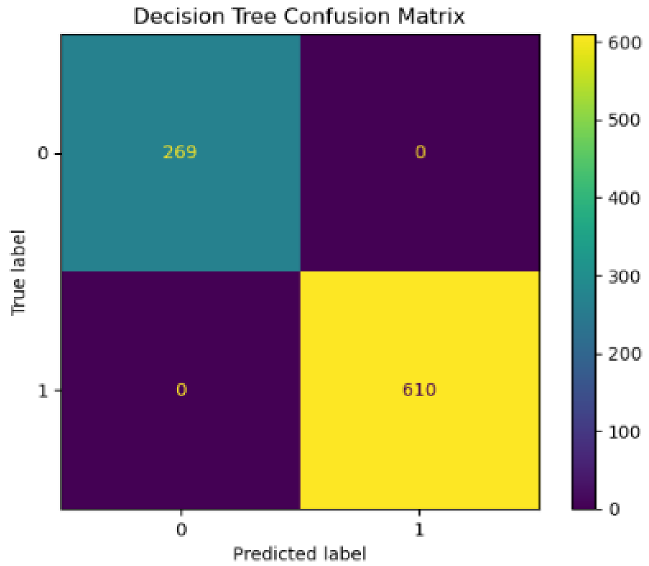
```
logreg = LogisticRegression(max_iter=1000, random_state=42)
logreg.fit(X_train_sm, y_train_sm)
evaluate_model(logreg, X_test, y_test, "Logistic Regression")
```

```
--- Logistic Regression Performance ---
              precision    recall  f1-score   support

           0       0.45      0.65      0.53       269
           1       0.81      0.64      0.72       610

    accuracy                           0.65       879
   macro avg       0.63      0.65      0.62       879
weighted avg       0.70      0.65      0.66       879
```




```
dtree = DecisionTreeClassifier(random_state=42)
dtree.fit(X_train_sm, y_train_sm)
evaluate_model(dtree, X_test, y_test, "Decision Tree")
```

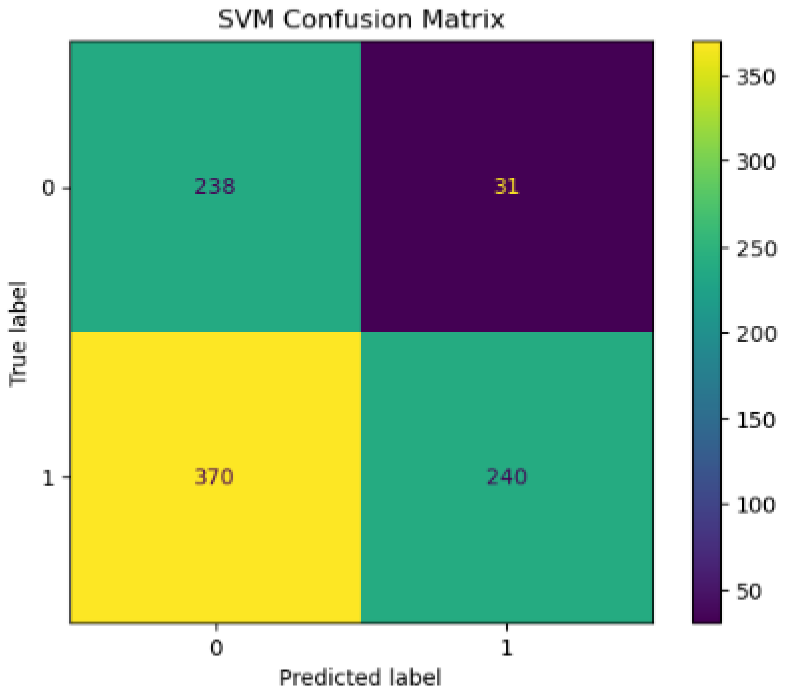
```
--- Decision Tree Performance ---
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       269
           1       1.00      1.00      1.00       610

    accuracy                           1.00       879
   macro avg       1.00      1.00      1.00       879
weighted avg       1.00      1.00      1.00       879
```




```
Accuracy: 1.0000
```

```
svm = SVC(kernel='rbf', probability=True, random_state=42)
svm.fit(X_train_sm, y_train_sm)
evaluate_model(svm, X_test, y_test, "SVM")
```

```
--- SVM Performance ---
              precision    recall  f1-score   support

           0       0.39      0.88      0.54       269
           1       0.89      0.39      0.54       610

    accuracy                           0.54       879
   macro avg       0.64      0.64      0.54       879
weighted avg       0.73      0.54      0.54       879
```




```
Accuracy: 0.5438
```

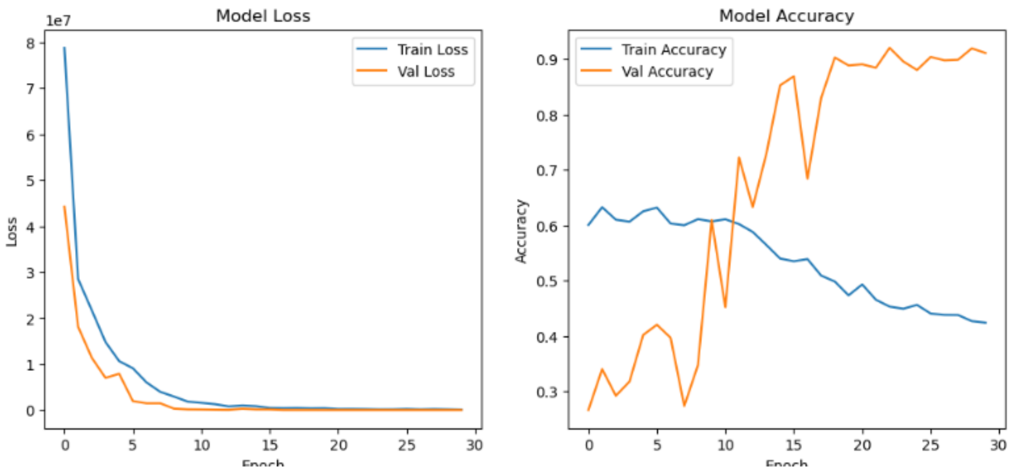
Advanced: Neural Network Using Keras/TensorFlow

- Build and train deep learning models
- Monitor training with loss and accuracy plots

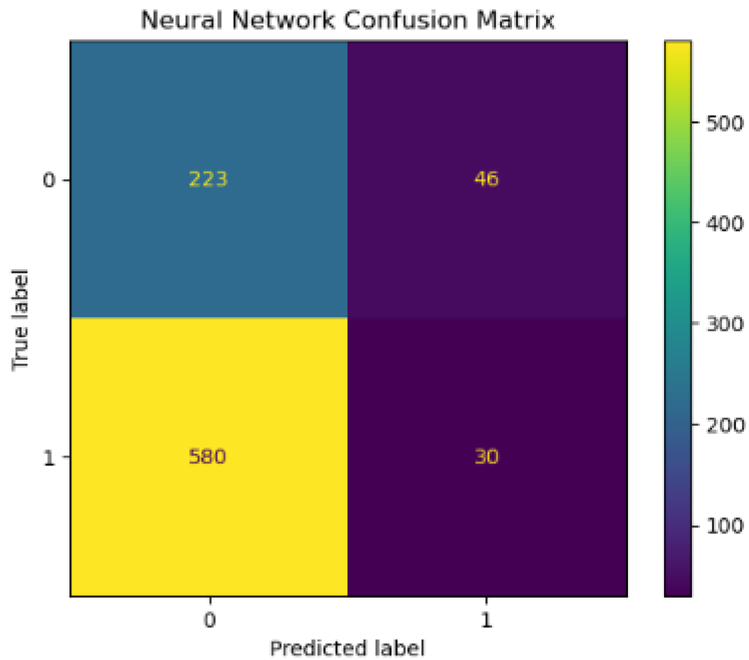
```
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_sm.shape[1],)),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

history = model.fit(
    X_train_sm, y_train_sm,
    epochs=50, batch_size=64,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=2
)
```

```
Epoch 1/50
61/61 - 7s - 117ms/step - accuracy: 0.6005 - loss: 78809192.0000 - val_accuracy: 0.2667 - val_loss: 44236824.0000
Epoch 2/50
61/61 - 1s - 15ms/step - accuracy: 0.6325 - loss: 28506814.0000 - val_accuracy: 0.3405 - val_loss: 18113800.0000
Epoch 3/50
61/61 - 1s - 15ms/step - accuracy: 0.6100 - loss: 21699984.0000 - val_accuracy: 0.2923 - val_loss: 11364683.0000
Epoch 4/50
61/61 - 1s - 15ms/step - accuracy: 0.6064 - loss: 14798091.0000 - val_accuracy: 0.3179 - val_loss: 6985701.5000
Epoch 5/50
61/61 - 1s - 15ms/step - accuracy: 0.6251 - loss: 10627766.0000 - val_accuracy: 0.4021 - val_loss: 7880368.0000
Epoch 6/50
61/61 - 1s - 24ms/step - accuracy: 0.6318 - loss: 9080753.0000 - val_accuracy: 0.4205 - val_loss: 1934074.1250
Epoch 7/50
61/61 - 1s - 20ms/step - accuracy: 0.6030 - loss: 5991019.0000 - val_accuracy: 0.3969 - val_loss: 1458984.0000
Epoch 8/50
61/61 - 1s - 16ms/step - accuracy: 0.5999 - loss: 3947458.7500 - val_accuracy: 0.2738 - val_loss: 1459792.5000
Epoch 9/50
61/61 - 1s - 16ms/step - accuracy: 0.6110 - loss: 2924424.0000 - val_accuracy: 0.3477 - val_loss: 296646.9062
Epoch 10/50
61/61 - 1s - 15ms/step - accuracy: 0.6069 - loss: 1826563.6250 - val_accuracy: 0.6092 - val_loss: 133943.5938
Epoch 11/50
61/61 - 1s - 14ms/step - accuracy: 0.6107 - loss: 1559884.5000 - val_accuracy: 0.4523 - val_loss: 108601.4141
Epoch 12/50
61/61 - 1s - 14ms/step - accuracy: 0.6020 - loss: 1263774.3750 - val_accuracy: 0.7221 - val_loss: 60828.4180
Epoch 13/50
61/61 - 1s - 15ms/step - accuracy: 0.5879 - loss: 785471.9375 - val_accuracy: 0.6328 - val_loss: 25745.7520
Epoch 14/50
61/61 - 1s - 13ms/step - accuracy: 0.5643 - loss: 979859.4375 - val_accuracy: 0.7292 - val_loss: 279165.2500
Epoch 15/50
61/61 - 1s - 14ms/step - accuracy: 0.5399 - loss: 828754.4375 - val_accuracy: 0.8523 - val_loss: 133465.9219
Epoch 16/50
61/61 - 1s - 15ms/step - accuracy: 0.5350 - loss: 438480.0000 - val_accuracy: 0.8687 - val_loss: 122314.2188
Epoch 17/50
61/61 - 1s - 17ms/step - accuracy: 0.5389 - loss: 390706.7188 - val_accuracy: 0.6841 - val_loss: 6708.6865
Epoch 18/50
61/61 - 1s - 16ms/step - accuracy: 0.5091 - loss: 425049.8750 - val_accuracy: 0.8297 - val_loss: 2177.5488
Epoch 19/50
61/61 - 1s - 17ms/step - accuracy: 0.4983 - loss: 353946.9688 - val_accuracy: 0.9026 - val_loss: 2164.8479
Epoch 20/50
61/61 - 1s - 17ms/step - accuracy: 0.4734 - loss: 384174.8438 - val_accuracy: 0.8882 - val_loss: 603.6735
Epoch 21/50
61/61 - 1s - 19ms/step - accuracy: 0.4932 - loss: 201253.5000 - val_accuracy: 0.8903 - val_loss: 909.3901
Epoch 22/50
```




```python
y_pred_prob_nn = model.predict(X_test).ravel()
y_pred_nn = (y_pred_prob_nn >= 0.5).astype(int)

print("\nNeural Network Classification Report:")
print(classification_report(y_test, y_pred_nn))

cm_nn = confusion_matrix(y_test, y_pred_nn)
ConfusionMatrixDisplay(cm_nn).plot()
plt.title('Neural Network Confusion Matrix')
plt.show()

print(f"Neural Network Accuracy: {accuracy_score(y_test, y_pred_nn):.4f}")
```




```
Neural Network Accuracy: 0.2878
```

Summary of Project

This project focused on sorting stock market data into different categories using machine learning. The data included prices, trading volumes, and other market indicators. We cleaned the data, fixed missing values, and balanced the different groups so the model could learn better. Then, we trained different models, including a neural network, to classify the stocks. The results were good and show that such models can help investors understand the market better. With more data and tuning, the models could become even more accurate.