

```
## Data Set (https://www.kaggle.com/datasets/kyanyoga/sample-sales-data)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
df = pd.read_csv('/content/drive/MyDrive/MLProject1/sales_data_sample.csv', encoding='unicode_escape')
```

```
df.head()
```

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	ADDRESS1
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped	1	2	2003	...	897 Airport Av
1	10121	34	81.35	5	2765.90	5/7/2003 0:00	Shipped	2	5	2003	...	59 n l'At
2	10134	41	94.74	2	3884.34	7/1/2003 0:00	Shipped	3	7	2003	...	27 n Colonel F
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped	3	8	2003	...	78934 Hi
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped	4	10	2003	...	7734 S

```
5 rows × 25 columns
```

```
df.shape
```

```
(2823, 25)
```

```
#####
##   Handle missing values   ##
#####
```

```
# 1. Inspect missing values
# -----
print("Missing values per column:\n", df.isnull().sum())
```

```
Missing values per column:
ORDERNUMBER      0
QUANTITYORDERED  0
PRICEEACH        0
ORDERLINENUMBER  0
SALES            0
ORDERDATE        0
STATUS           0
QTR_ID           0
MONTH_ID         0
```

```
YEAR_ID          0
PRODUCTLINE      0
MSRP             0
PRODUCTCODE      0
CUSTOMERNAME     0
PHONE            0
ADDRESSLINE1     0
ADDRESSLINE2     2521
CITY             0
STATE           1486
POSTALCODE       76
COUNTRY          0
TERRITORY        1074
CONTACTLASTNAME  0
CONTACTFIRSTNAME 0
DEALSIZE         0
dtype: int64
```

```
# 2. Drop columns with too many missing values (ADDRESSLINE2)
if "ADDRESSLINE2" in df.columns:
    if df["ADDRESSLINE2"].isnull().mean() > 0.5:
        df = df.drop(columns=["ADDRESSLINE2"])

# 3. Fill missing values
# -----

# Fill missing categorical values with mode
categorical_cols = df.select_dtypes(include=['object']).columns

for col in categorical_cols:
    df[col] = df[col].fillna(df[col].mode()[0])

# Fill missing numerical values with median
numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns

for col in numeric_cols:
    df[col] = df[col].fillna(df[col].median())

# 4. Verify no missing values remain
# -----
print("\nMissing values after cleaning:\n", df.isnull().sum())
```

```
Missing values after cleaning:
ORDERNUMBER      0
QUANTITYORDERED  0
PRICEEACH        0
ORDERLINENUMBER  0
SALES            0
ORDERDATE        0
STATUS           0
QTR_ID           0
MONTH_ID         0
YEAR_ID          0
PRODUCTLINE      0
MSRP             0
PRODUCTCODE      0
CUSTOMERNAME     0
PHONE            0
ADDRESSLINE1     0
CITY             0
STATE            0
POSTALCODE       0
COUNTRY          0
TERRITORY        0
CONTACTLASTNAME  0
CONTACTFIRSTNAME 0
DEALSIZE         0
dtype: int64
```

```
#Save cleaned dataset
df.to_csv("sales_data_cleaned.csv", index=False)

print("\nDataset cleaned and saved as sales_data_cleaned.csv")
```

```
Dataset cleaned and saved as sales_data_cleaned.csv
```

```
#####
### Apply machine learning models using a grid-search optimizer on the cleaned dataset, and evaluate the performance of the mod
#####

# 1. Load cleaned dataset
# -----
df = pd.read_csv('sales_data_cleaned.csv')
# 2. Select target and features
# -----
target = "SALES"

# Drop date & text-heavy columns not useful for numeric ML
drop_cols = ["ORDERDATE", "CUSTOMERNAME", "PHONE", "CITY",
             "CONTACTLASTNAME", "CONTACTFIRSTNAME", "ADDRESSLINE1",
             "POSTALCODE", "STATE"]

df = df.drop(columns=[c for c in drop_cols if c in df.columns])

X = df.drop(columns=[target])
y = df[target]

# 3. Identify categorical & numerical columns
# -----
categorical_cols = X.select_dtypes(include=['object']).columns
numeric_cols = X.select_dtypes(include=['int64', 'float64']).columns

# Encoding for categorical features
preprocess = ColumnTransformer(
    transformers=[
        ("cat", OneHotEncoder(handle_unknown="ignore"), categorical_cols),
        ("num", "passthrough", numeric_cols)
    ]
)

# 4. Train/test split
# -----
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=42
)

# 5. Define ML models + Grid Search parameters
# -----
models = {
    "RandomForest": {
        "pipeline": Pipeline([
            ("preprocess", preprocess),
            ("model", RandomForestRegressor())
        ]),
        "params": {
            "model__n_estimators": [100, 200],
            "model__max_depth": [10, 20, None],
            "model__min_samples_split": [2, 5]
        }
    },
    "LinearRegression": {
        "pipeline": Pipeline([
            ("preprocess", preprocess),
            ("model", LinearRegression())
        ]),
        "params": {
            "model__fit_intercept": [True, False],
            "model__normalize": [True, False]
        }
    }
}

# 6. Run Grid Search for each model
# -----
results = {}

if "LinearRegression" in models and "model__normalize" in models["LinearRegression"]["params"]:
    del models["LinearRegression"]["params"]["model__normalize"]

for name, mp in models.items():
    print(f"\n Running GridSearchCV for {name}...")
```

```

gs = GridSearchCV(
    estimator=mp["pipeline"],
    param_grid=mp["params"],
    cv=5,
    scoring="r2",
    n_jobs=-1
)

gs.fit(X_train, y_train)

best_model = gs.best_estimator_
y_pred = best_model.predict(X_test)

r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

results[name] = {
    "Best Params": gs.best_params_,
    "R2 Score": r2,
    "RMSE": rmse
}

print(f"Best Params: {gs.best_params_}")
print(f"R2 Score: {r2:.4f}")
print(f"RMSE: {rmse:.2f}")

```

```

Running GridSearchCV for RandomForest...
Best Params: {'model__max_depth': 10, 'model__min_samples_split': 5, 'model__n_estimators': 100}
R2 Score: 0.9223
RMSE: 582.25

Running GridSearchCV for LinearRegression...
Best Params: {'model__fit_intercept': True}
R2 Score: 0.8925
RMSE: 685.09

```

```

# 7. Display final comparison results
# -----
print("\n📊 FINAL MODEL PERFORMANCE")
for model, info in results.items():
    print(f"\nModel: {model}")
    print(f"    • Best Params: {info['Best Params']}")
    print(f"    • R2 Score: {info['R2 Score']:.4f}")
    print(f"    • RMSE: {info['RMSE']:.2f}")

```

📊 FINAL MODEL PERFORMANCE

```

Model: RandomForest
    • Best Params: {'model__max_depth': 10, 'model__min_samples_split': 2, 'model__n_estimators': 100}
    • R2 Score: 0.9204
    • RMSE: 589.30

Model: LinearRegression
    • Best Params: {'model__fit_intercept': True}
    • R2 Score: 0.8925
    • RMSE: 685.09

```

```

#####
#### Use two different feature-selection techniques to identify the most important features in the dataset #####
#####
# Drop text-heavy or irrelevant columns
drop_cols = ["ORDERDATE", "CUSTOMERNAME", "PHONE", "CITY",
             "CONTACTLASTNAME", "CONTACTFIRSTNAME", "ADDRESSLINE1",
             "POSTALCODE", "STATE"]

df = df.drop(columns=[c for c in drop_cols if c in df.columns])

X = df.drop(columns=[target])
y = df[target]

# Identify categorical and numeric columns
categorical_cols = X.select_dtypes(include=['object']).columns
numeric_cols = X.select_dtypes(include=['int64', 'float64']).columns

```

```

# Preprocessing: OneHotEncode categorical variables
preprocess = ColumnTransformer(
    transformers=[
        ("cat", OneHotEncoder(handle_unknown="ignore"), categorical_cols),
        ("num", "passthrough", numeric_cols),
    ],
    remainder="drop"
)

# Fit transform
X_encoded = preprocess.fit_transform(X)

# Feature names after encoding
feature_names = preprocess.get_feature_names_out()
# -----
# METHOD 1: Mutual Information Regression (MI)
# -----
print("\n🔍 FEATURE SELECTION METHOD 1 – Mutual Information (MI)\n")

from sklearn.feature_selection import mutual_info_regression

mi_scores = mutual_info_regression(X_encoded, y, random_state=42)

mi_series = pd.Series(mi_scores, index=feature_names).sort_values(ascending=False)

print(mi_series.head(15)) # Show top 15 important features

```

🔍 FEATURE SELECTION METHOD 1 – Mutual Information (MI)

cat__DEALSIZE_Medium	0.692197
cat__DEALSIZE_Small	0.688678
num__PRICEEACH	0.412934
num__MSRP	0.377462
cat__DEALSIZE_Large	0.214398
num__QUANTITYORDERED	0.192991
num__QTR_ID	0.022195
cat__PRODUCTLINE_Classic Cars	0.019218
cat__PRODUCTLINE_Vintage Cars	0.016641
cat__PRODUCTLINE_Ships	0.013385
cat__PRODUCTCODE_S24_2840	0.012026
cat__PRODUCTCODE_S24_1937	0.011943
cat__PRODUCTCODE_S24_3969	0.011734
cat__PRODUCTCODE_S24_2972	0.011712
cat__PRODUCTCODE_S18_3232	0.011607

dtype: float64

```

# Feature Selection Method 1:
#   Model-Based Importance using RandomForest
# -----

# Preprocessing for RF (OneHotEncoding)
preprocess = ColumnTransformer(
    transformers=[
        ("cat", OneHotEncoder(handle_unknown="ignore"), categorical_cols),
        ("num", "passthrough", numeric_cols)
    ]
)

X_encoded = preprocess.fit_transform(X)
feature_names = preprocess.get_feature_names_out()

# Train Random Forest to compute importance
rf = RandomForestRegressor(n_estimators=300, random_state=42)
rf.fit(X_encoded, y)

rf_importances = pd.Series(rf.feature_importances_, index=feature_names)
rf_top15 = rf_importances.sort_values(ascending=False).head(15)

print("\n🔍 MODEL-BASED FEATURE IMPORTANCE (TOP 15):\n")
print(rf_top15)

```

🔍 MODEL-BASED FEATURE IMPORTANCE (TOP 15):

cat__DEALSIZE_Small	0.546543
cat__DEALSIZE_Large	0.150528
num__QUANTITYORDERED	0.092392
cat__DEALSIZE_Medium	0.072838

```

num__PRICEEACH          0.067695
num__MSRP               0.030126
num__ORDERNUMBER        0.009390
num__ORDERLINENUMBER    0.005075
num__MONTH_ID           0.003686
cat__PRODUCTCODE_S18_2325 0.001358
cat__COUNTRY_USA        0.000973
num__QTR_ID             0.000934
cat__COUNTRY_France     0.000741
cat__PRODUCTLINE_Vintage Cars 0.000672
num__YEAR_ID            0.000633
dtype: float64

```

```

# Feature Selection Method 2:
#   SelectKBest (F-test)
# -----
from sklearn.feature_selection import SelectKBest, f_regression

selector = SelectKBest(score_func=f_regression, k=15)
selector.fit(X[numeric_cols], y) # SelectKBest works only on numeric features

f_scores = pd.Series(selector.scores_, index=numeric_cols).sort_values(ascending=False)
f_top15 = f_scores.head(15)

print("\n🔍 SELECTKBEST (F-TEST) TOP 15 FEATURES:\n")
print(f_top15)

```

🔍 SELECTKBEST (F-TEST) TOP 15 FEATURES:

```

PRICEEACH          2152.157001
MSRP               1908.485013
QUANTITYORDERED    1232.573529
ORDERLINENUMBER     9.654243
ORDERNUMBER         4.502577
YEAR_ID            3.589208
MONTH_ID           0.260288
QTR_ID             0.130299
dtype: float64

```

```

#####
#####   Apply machine learning models again using only the selected features, and evaluate the results   #####
#####

# 1. Preprocess (OneHotEncode categorical variables)
# -----
preprocess = ColumnTransformer(
    transformers=[
        ("cat", OneHotEncoder(handle_unknown="ignore"), categorical_cols),
        ("num", "passthrough", numeric_cols)
    ]
)

# Fit & encode features
X_encoded = preprocess.fit_transform(X)
feature_names = preprocess.get_feature_names_out()

# 2. Feature Selection – Model-Based Importance
# -----
rf = RandomForestRegressor(n_estimators=300, random_state=42)
rf.fit(X_encoded, y)

importances = pd.Series(rf.feature_importances_, index=feature_names)
selected_features = importances.sort_values(ascending=False).head(15).index.tolist()

print("\nSelected Top 15 Features:")
for f in selected_features:
    print(" •", f)

# Reduce X_encoded to selected features only
selected_idx = [list(feature_names).index(f) for f in selected_features]
X_selected = X_encoded[:, selected_idx]

# 3. Train/Test Split
# -----
X_train, X_test, y_train, y_test = train_test_split(
    X_selected, y, test_size=0.20, random_state=42
)

```

```
# -----
# 4. Define ML Models + GridSearchCV
# -----

models = {
    "RandomForest": {
        "model": RandomForestRegressor(),
        "params": {
            "n_estimators": [100, 200],
            "max_depth": [None, 10, 20],
            "min_samples_split": [2, 5]
        }
    },

    "LinearRegression": {
        "model": LinearRegression(),
        "params": {
            "fit_intercept": [True, False]
        }
    }
}

results = {}
```

Selected Top 15 Features:

- cat_DEALSIZE_Small
- cat_DEALSIZE_Large
- num_QUANTITYORDERED
- cat_DEALSIZE_Medium
- num_PRICEEACH
- num_MSRP
- num_ORDERNUMBER
- num_ORDERLINENUMBER
- num_MONTH_ID
- cat_PRODUCTCODE_S18_2325
- cat_COUNTRY_USA
- num_QTR_ID
- cat_COUNTRY_France
- cat_PRODUCTLINE_Vintage Cars
- num_YEAR_ID

```
# 5. Train models using ONLY selected features
# -----

for name, mp in models.items():
    print(f"\nTraining {name} using selected features...")

    gs = GridSearchCV(
        estimator=mp["model"],
        param_grid=mp["params"],
        cv=5,
        scoring="r2",
        n_jobs=-1
    )

    gs.fit(X_train, y_train)
    best_model = gs.best_estimator_

    y_pred = best_model.predict(X_test)
    r2 = r2_score(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))

    results[name] = {"Best Params": gs.best_params_, "R2": r2, "RMSE": rmse}

    print("Best Parameters:", gs.best_params_)
    print("R² Score:", r2)
    print("RMSE:", rmse)
```

Training RandomForest using selected features...
 Best Parameters: {'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 200}
 R² Score: 0.9279087201037138
 RMSE: 560.9550055227952

Training LinearRegression using selected features...
 Best Parameters: {'fit_intercept': True}
 R² Score: 0.8838897648252394
 RMSE: 711.9047850361341

```
#6. Final Results Summary
# -----
print("\n\n📊 Final Model Performance Using Only Selected Features:")
for model, info in results.items():
    print(f"\nModel: {model}")
    print("    Best Params:", info["Best Params"])
    print(f"    R² Score: {info['R2']:.4f}")
    print(f"    RMSE: {info['RMSE']:.2f}")
```

📊 Final Model Performance Using Only Selected Features:

```
Model: RandomForest
Best Params: {'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 200}
R² Score: 0.9279
RMSE: 560.96

Model: LinearRegression
Best Params: {'fit_intercept': True}
R² Score: 0.8839
RMSE: 711.90
```

FOR MY REFERENCE

##1. Classification Models

Used when the output is a category (e.g., Fraud / Not Fraud, Small / Medium / Large).

```
## Common classification models:
    ## 1- Logistic Regression
    ## 2- Decision Tree
    ## 3- Random Forest
    ## 4- Support Vector Machine (SVM)
    ## 5- K-Nearest Neighbors (KNN)
    ## 6- Naive Bayes
```

##2. Regression Models

Used when the output is a number (e.g., sales amount, price prediction).

```
## Common regression models:
    ## Linear Regression
    ## Random Forest Regression
    ## Support Vector Regression (SVR)
```

##3. Clustering Models

Used to group similar data without labels (unsupervised learning). Examples:

```
    ## K-Means
    ## Hierarchical Clustering
```

##4. Neural Networks

Advanced models inspired by the human brain. Examples:

```
    ##1- Artificial Neural Networks (ANNs)
    ##2- Convolutional Neural Networks (CNNs) → image tasks
    ##3- Recurrent Neural Networks (RNNs) → time series, text
```