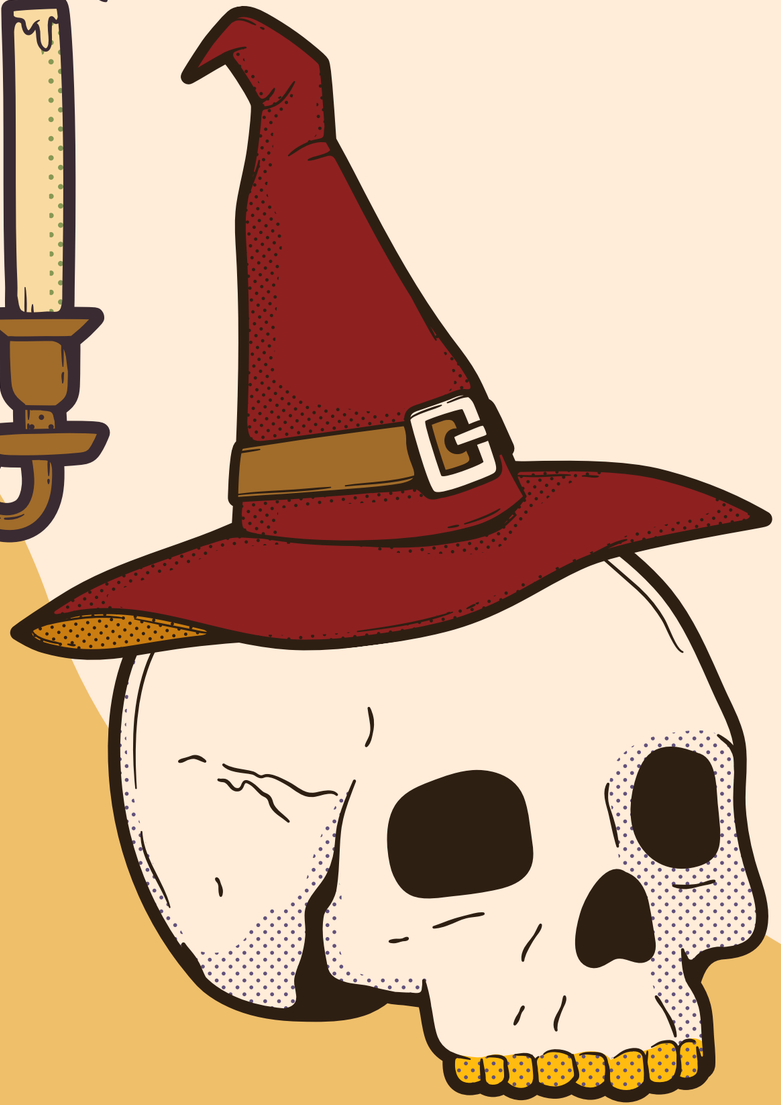
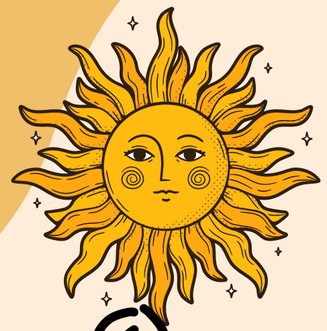


# Group Project

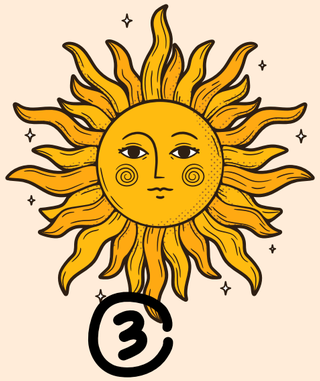
NINJAS

# Topic Outline



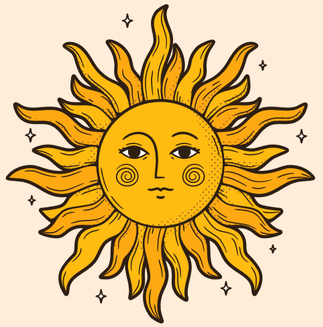
1. Dataset used

2. EDA explanation



3. preprocessing

4. Regression



5. Classification

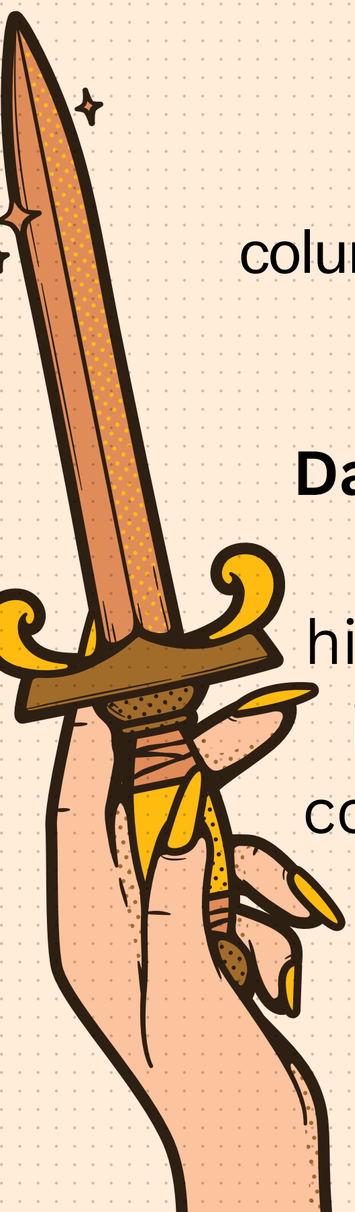
6. Pipeline

# Dataset

Dataset: "Consumer goods"

columns: 12 columns

rows: 15117 rows



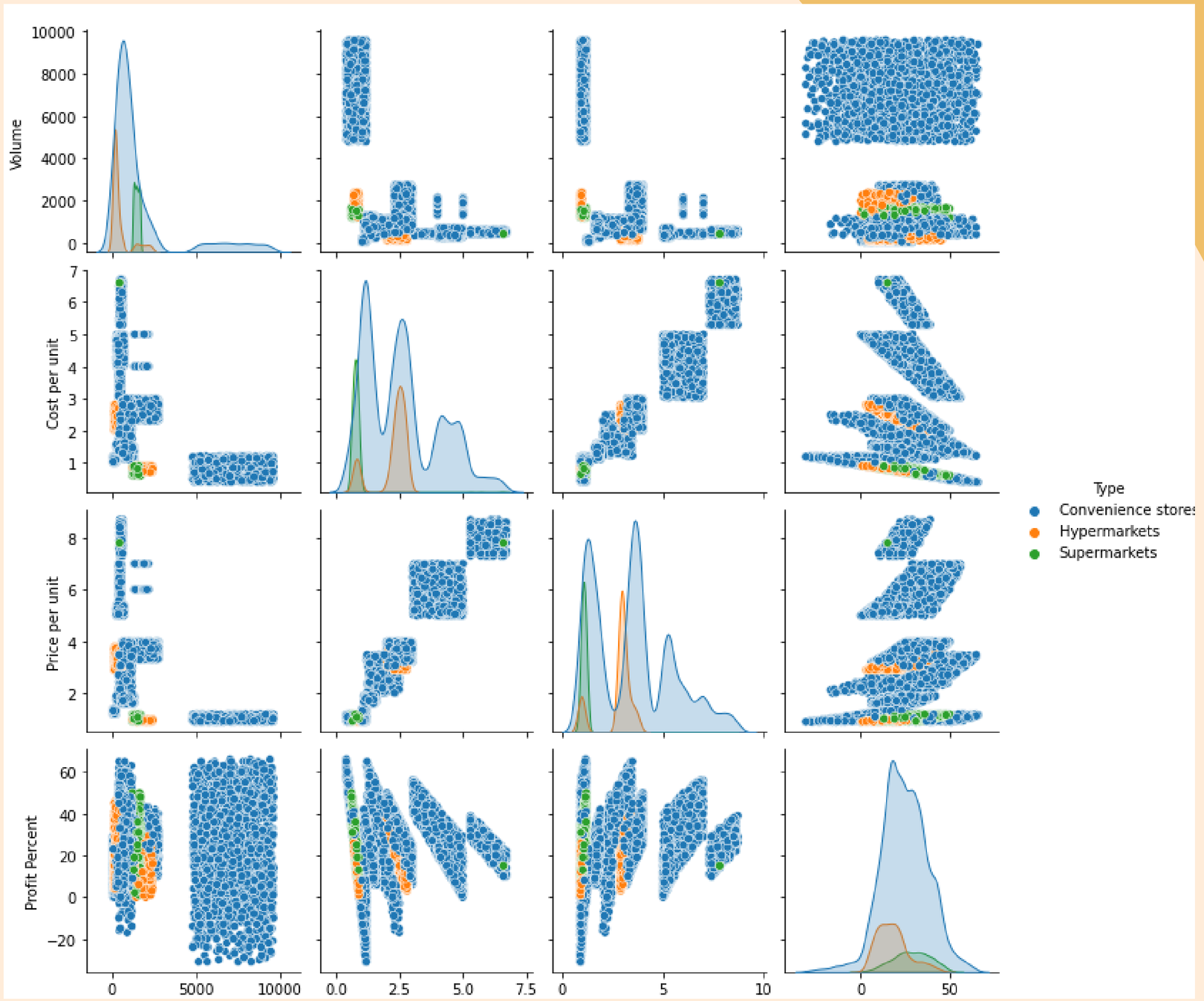
**Dataset domain:**

This dataset contains a lot of historical sales data. In general, it is about the types of products consumed and its producer. Also, costs and prices per unit and revenues with profits details.

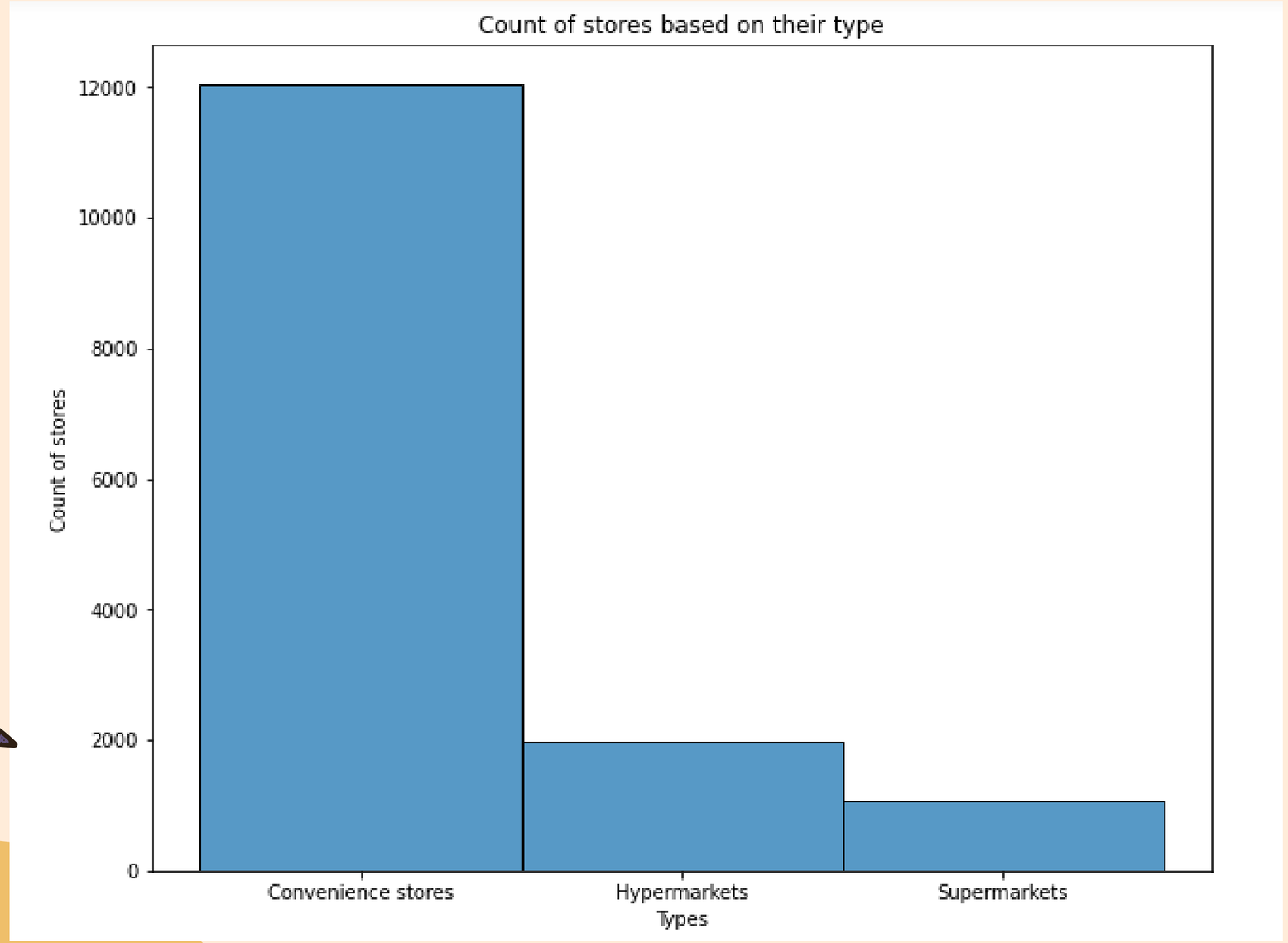
```
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Year             15117 non-null  object
 1   Type             15117 non-null  object
 2   Product group    15117 non-null  object
 3   Producer         15117 non-null  object
 4   Code             15117 non-null  object
 5   Volume           15117 non-null  int64
 6   Cost per unit    15117 non-null  float64
 7   Price per unit   15117 non-null  float64
 8   Revenues         15117 non-null  object
 9   Total Cost       15117 non-null  object
 10  Profit Percent   15117 non-null  int64
 11  Summary Profits  15116 non-null  object
```

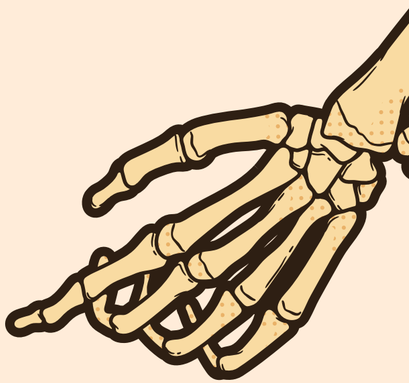

# EDA explanation

# PAIR PLOT

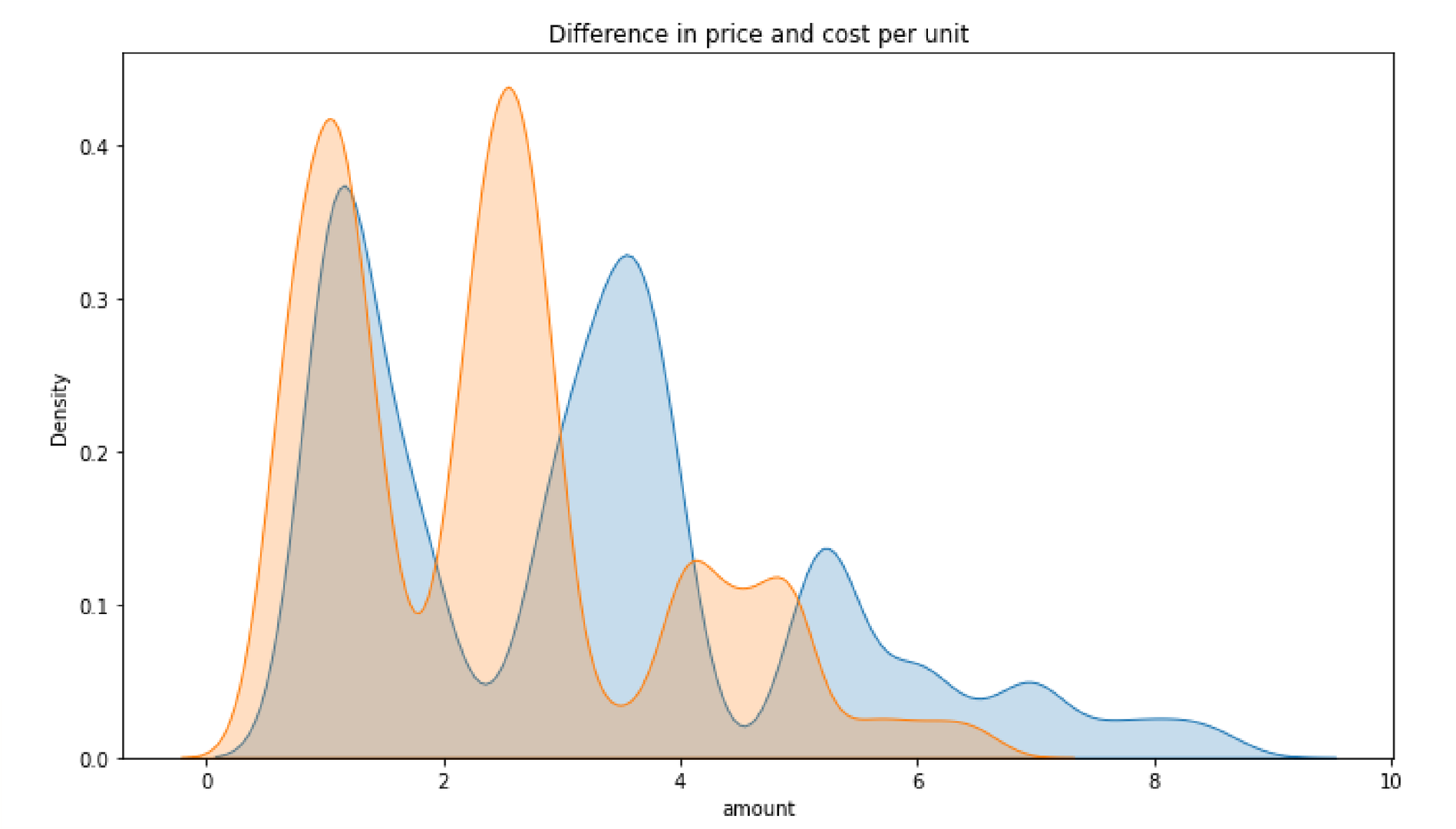

Volume
Cost per unit
Price per unit
Profit Percent
Type
Convenience stores
Hypermarkets
Supermarkets

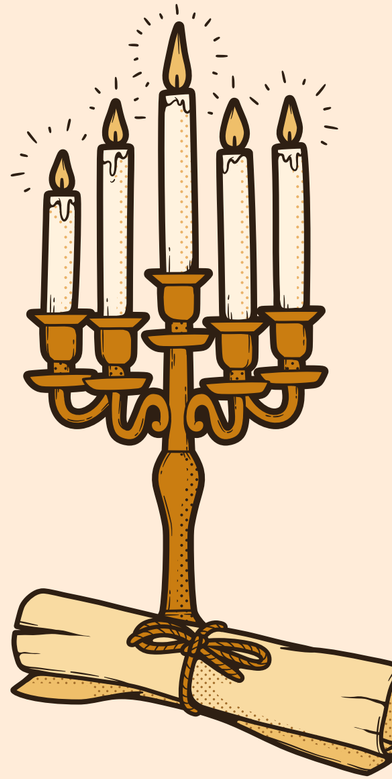
# 2.count of stores based on their type



# 03 | difference in price and cost per unit



# cleaning data

## dropping the null data and unnecessary columns and changing the data type

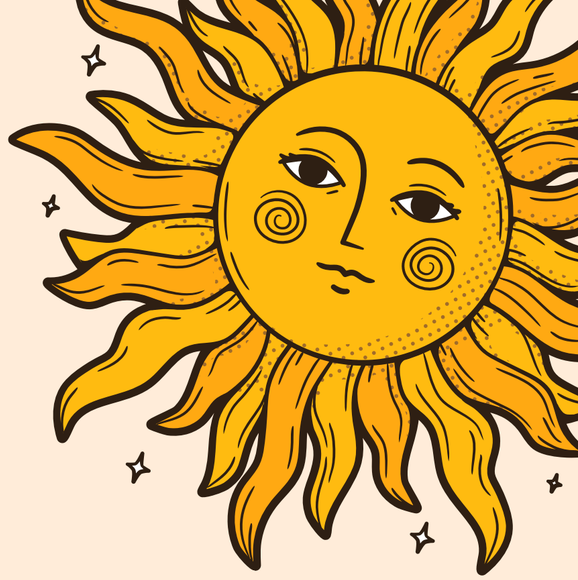
```
#DROPPING THE NULL DATA

df['Summary Profits'].fillna(df['Summary Profits'].mode()[0], inplace = True)
```

```
df=df.drop(['Year','Code'],axis=1)
df.head()
```

| | Type | Product group | Producer | Volume | Cost per unit | Price per unit | Revenues | Total Cost | Profit Percent | Summary Profits |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Convenience stores | Meat | J&F | 420 | 6.00 | 8.40 | $3,542.00 | $2,528.40 | 29 | Good Profit |
| 1 | Convenience stores | Meat | J&F | 480 | 6.10 | 7.40 | $3,558.40 | $2,947.20 | 17 | Acceptable Profit |
| 2 | Convenience stores | Meat | J&F | 528 | 5.40 | 7.60 | $4,019.84 | $2,837.12 | 29 | Good Profit |
| 3 | Convenience stores | Coffee | J&F | 4836 | 0.74 | 0.96 | $4,642.56 | $3,578.64 | 23 | Good Profit |
| 4 | Convenience stores | Coffee | J&F | 5928 | 0.95 | 0.99 | $5,868.72 | $5,631.60 | 4 | Acceptable Profit |

```
df['Total Cost'] = df['Total Cost'].str.replace('$','')
df['Total Cost'] = df['Total Cost'].str.replace(',','')
df['Total Cost'] = df['Total Cost'].astype(float)
```

```
df['Revenues'] = df['Revenues'].str.replace('$','')
df['Revenues'] = df['Revenues'].str.replace(',','')
df['Revenues'] = df['Revenues'].astype(float)
```



# PREPROCESSING

# splitting the data

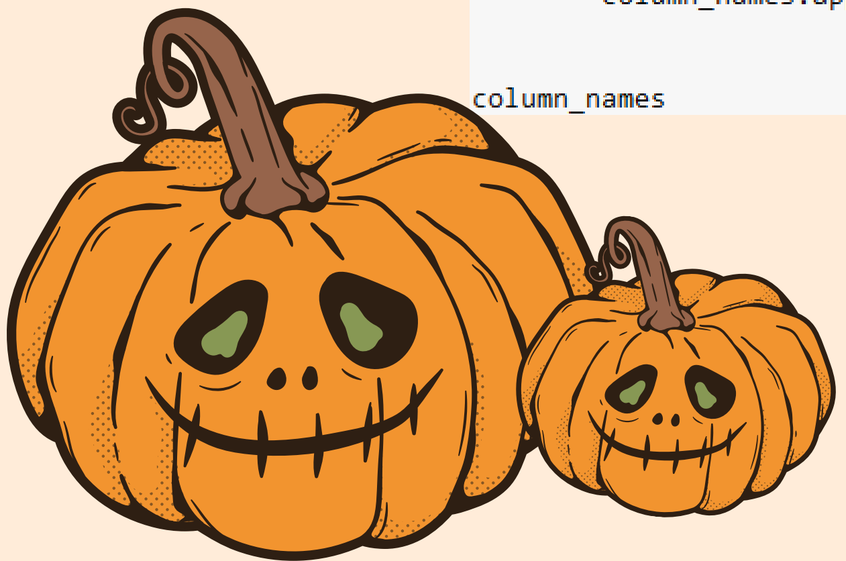
```
# Splitting data

# Split into training and test sets
training, test = train_test_split(
    df,
    train_size=0.8,
    test_size=0.2,
    random_state=42
)
```

# ONE HOT ENCODER

```
# Create our One Hot Encoder object
one_hot = OneHotEncoder()

col_names = ["Type", "Product group","Producer"]

# One Hot encode the column in both the train and validation sets
one_hot_df = one_hot.fit_transform(training[col_names]).toarray()

one_hot_df_val = one_hot.transform(test[col_names]).toarray()
```

```
# Look at the categories
one_hot.categories_
```

```
# Create column names list for one hot encoded features
column_names = []

for y in range(len(one_hot.categories_)):
    for z in range(len(one_hot.categories_[y])):
        # print(one_hot.categories_[y][z])
        column_names.append(col_names[y]+"_"+one_hot.categories_[y][z])


column_names
```



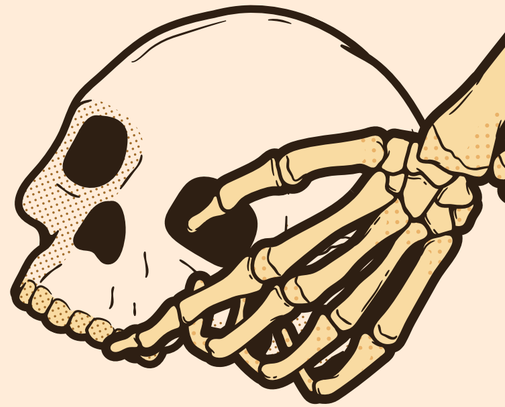
# ONE HOT ENCODER

```
oh_df = pd.DataFrame(
    one_hot_df,
    index=training.index,
    columns = column_names
)

oh_df_val = pd.DataFrame(
    one_hot_df_val,
    index=test.index,
    columns = column_names
)

print(oh_df.shape)
print(oh_df_val.shape)
```

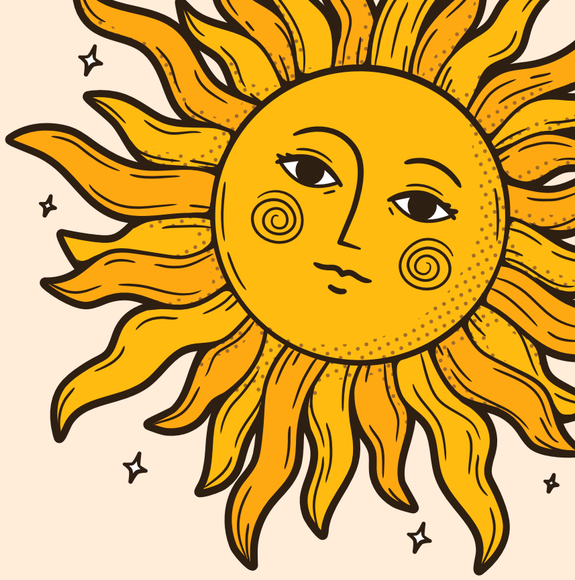
```
# Create train_new df by merging train and one hot encoded df together and drop color and clarity cols

training = training.merge(oh_df, on=training.index).set_index("key_0").drop(["Type", "Product group","Producer"], axis=1)
test = test.merge(oh_df_val, on=test.index).set_index("key_0").drop(["Type", "Product group","Producer"], axis=1)

print(training.shape)
print(test.shape)
```

# ORDINAL ENCODER

```
ordinal = OrdinalEncoder()

col_names = ["Summary Profits"]

# Ordinal encode the column
ordinal_ls = ordinal.fit_transform(training[col_names])
ordinal_ls_val = ordinal.transform(test[col_names])
# Look at categories
ordinal.categories_
# Add oridnally encoded column to the data
training["Summary Profits"] = ordinal_ls
test["Summary Profits"] = ordinal_ls_val

training.sample(5)
```



# Regression



# Our objective

Predict the

total cost

# Linear Regression

```
print('MAE:', metrics.mean_absolute_error(y_true=y_test, y_pred=preds_lin))
print('MSE:', metrics.mean_squared_error(y_true=y_test, y_pred=preds_lin))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_true=y_test, y_pred=preds_lin)))
```

```
MAE: 223.2661372145819
MSE: 101413.56080009248
RMSE: 318.45495882478025
```

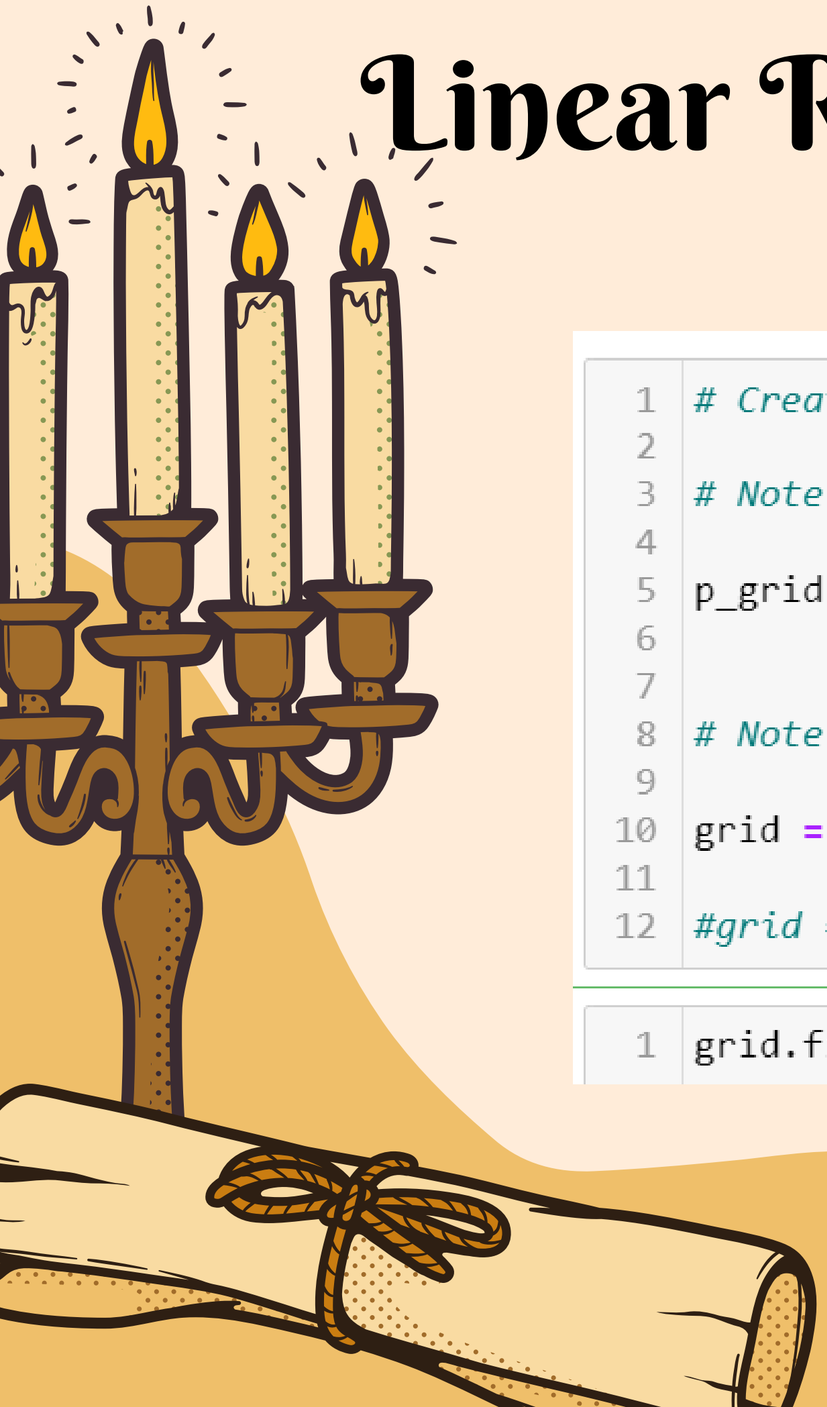
```
# Creating the feature set and the traget
#set for both the training and the validation

target = "Total Cost"

X_train = training.drop(target, axis=1)
y_train = training[target]

X_test = test.drop(target, axis=1)
y_test = test[target]
```

```
# Create a Pipeline for our model
pipe = make_pipeline(
    # 1st step handle missing values
    SimpleImputer(), # Impute missing values
    # scale columns
    StandardScaler(),
    # apply the model
    LinearRegression()
)

pipe.fit(X_train,y_train)
pipe.score(X_test, y_test)
```

```
0.9785356101077973
```

```
reg_lin = LinearRegression()
reg_lin.fit(X_train, y_train)
preds_lin = reg_lin.predict(X_test)
```

# Linear Regression - Tuning



```
# Create a dictiionary of the parameters you want to tune

# Note: more paramaters and values --> increase the run time

p_grid = {'fit_intercept': [True, False],
          'normalize': [True, False]}

# Note: by default the cross validation is set to 5

grid = GridSearchCV(lr, p_grid)

#grid = GridSearchCV(lr, p_grid, cv = 10)
```

```
grid.fit(X_train, y_train)
```

```
grid_df = pd.DataFrame(grid.cv_results_)

grid_df
```

| | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_fit_intercept | param_normalize | params | split0_test_score | split1_test_score | sp |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.309398 | 0.084541 | 0.019785 | 0.006453 | True | True | {'fit_intercept': True, 'normalize': True} | 0.978529 | 0.979049 | |
| **1** | 0.273971 | 0.095062 | 0.015148 | 0.005072 | True | False | {'fit_intercept': True, 'normalize': False} | 0.978522 | 0.979052 | |
| **2** | 0.330299 | 0.084729 | 0.020163 | 0.006058 | False | True | {'fit_intercept': False, 'normalize': True} | 0.978522 | 0.979052 | |
| **3** | 0.290131 | 0.104321 | 0.022215 | 0.004796 | False | False | {'fit_intercept': False, 'normalize': False} | 0.978522 | 0.979052 | |

```
parameters = ['param_fit_intercept', 'param_normalize', 'mean_test_score', 'rank_test_score']

grid_df[parameters]
```

| | param_fit_intercept | param_normalize | mean_test_score | rank_test_score |
|---|---|---|---|---|
| **0** | True | True | -7.105570e+19 | 4 |
| **1** | True | False | 9.776075e-01 | 3 |
| **2** | False | True | 9.776076e-01 | 1 |

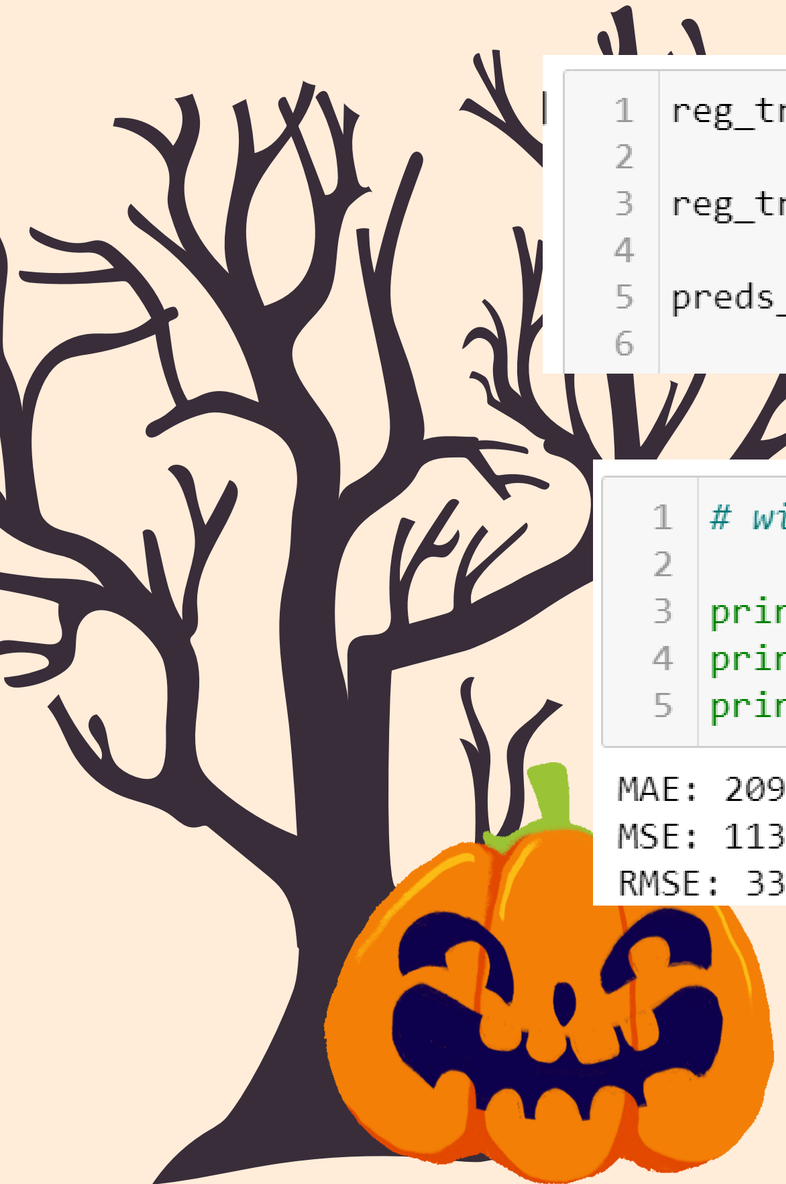
```
grid.best_score_
```

```
0.9776076340899342
```

```
print('MAE:', metrics.mean_absolute_error(y_true=y_test, y_pred=preds))
print('MSE:', metrics.mean_squared_error(y_true=y_test, y_pred=preds))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_true=y_test, y_pred=preds)))
```

```
MAE: 223.26613721441848
MSE: 101413.56080006914
RMSE: 318.4549588247436
```

# Decision Tree Regressor

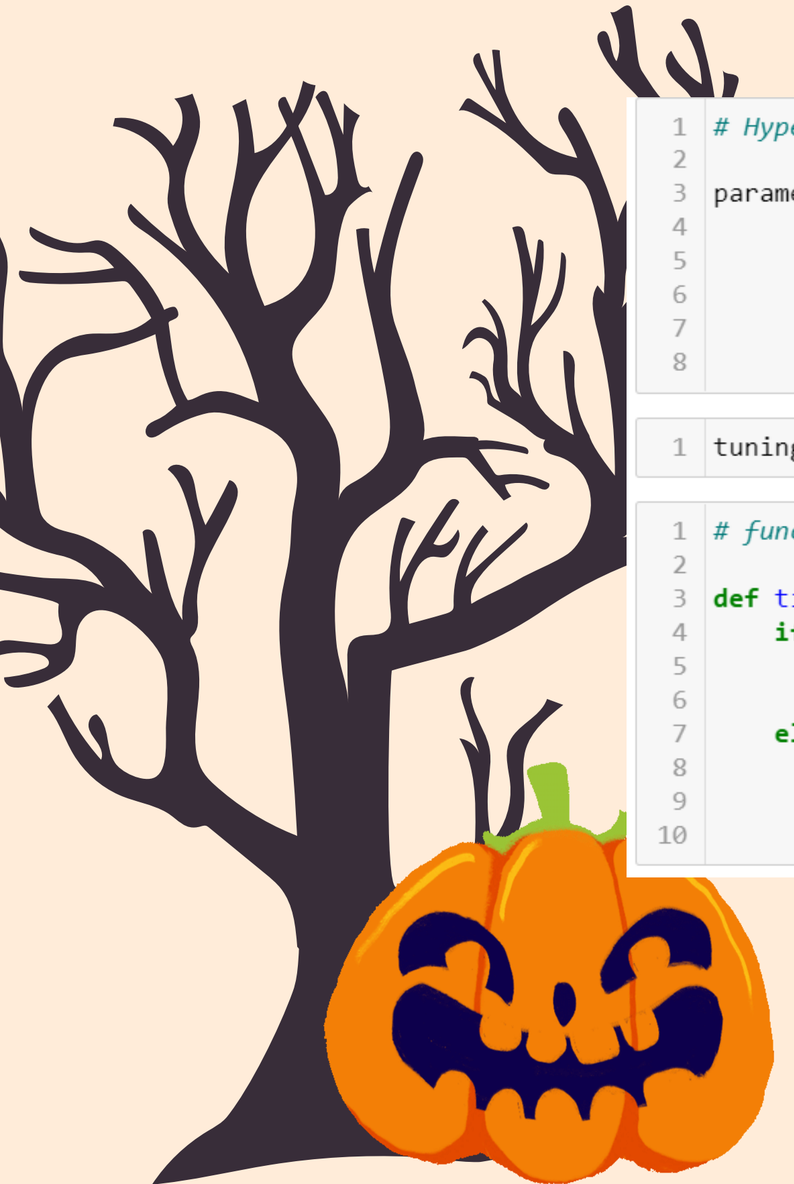


```python
reg_tree = DecisionTreeRegressor(random_state = 0, max_depth= 5, criterion= 'mse')

reg_tree.fit(X_train, y_train)

preds_tree = reg_tree.predict(X_test)

```

```python
# without hyperparameter tuning

print('MAE:', metrics.mean_absolute_error(y_true=y_test, y_pred=preds_tree))
print('MSE:', metrics.mean_squared_error(y_true=y_test, y_pred=preds_tree))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_true=y_test, y_pred=preds_tree)))
```

```
MAE: 209.16169570982345
MSE: 113367.47235318852
RMSE: 336.7008647942392
```

# Decision Tree Regressor - Tuning

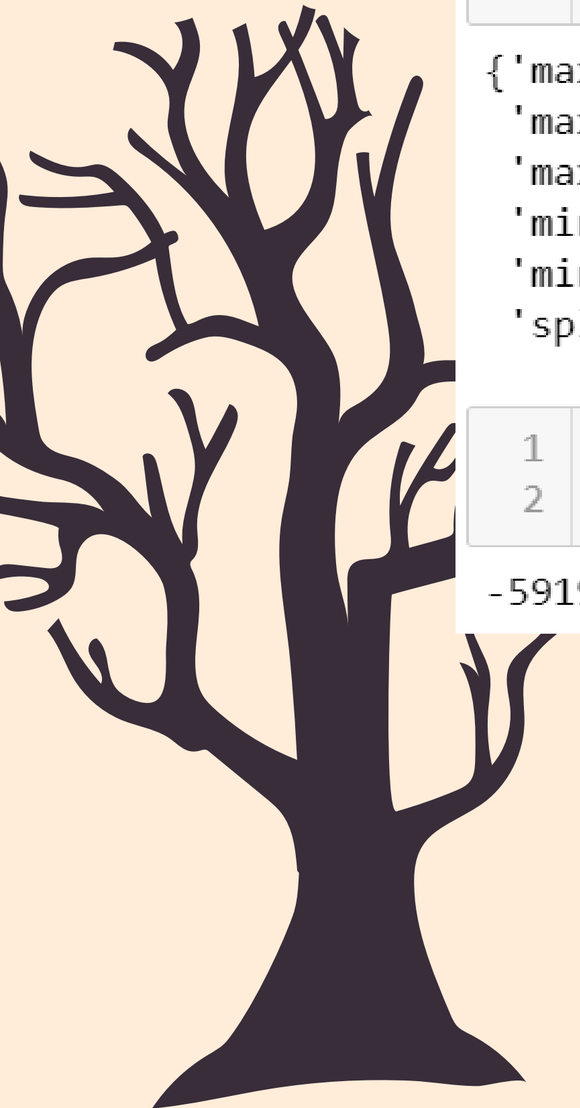


```python
# Hyper parameters range intialization for tuning

parameters={"splitter":["best","random"],
            "max_depth" : [1,3,5,7,9,11,12],
           "min_samples_leaf":[1,2,3,4,5,6,7,8,9,10],
           "min_weight_fraction_leaf":[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],
           "max_features":["auto","log2","sqrt",None],
           "max_leaf_nodes":[None,10,20,30,40,50,60,70,80,90] }
```

```python
tuning_model=GridSearchCV(reg_tree,param_grid=parameters,scoring='neg_mean_squared_error',cv=3,verbose=3)
```

```python
# function for calculating how much time take for hyperparameter tuning

def timer(start_time=None):
    if not start_time:
        start_time=datetime.now()
        return start_time
    elif start_time:
        thour,temp_sec=divmod((datetime.now()-start_time).total_seconds(),3600)
        tmin,tsec=divmod(temp_sec,60)
        print(thour,":",tmin,':',round(tsec,2))
```

# Decision Tree Regressor - Tuning



```
# best hyperparameters
tuning_model.best_params_
```

```
{'max_depth': 5,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'min_samples_leaf': 1,
 'min_weight_fraction_leaf': 0.1,
 'splitter': 'best'}
```

```
# best model score
tuning_model.best_score_
```

```
-591998.7883957286
```

```
print('MAE:', metrics.mean_absolute_error(y_test,tuned_pred))
print('MSE:', metrics.mean_squared_error(y_test, tuned_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, tuned_pred)))
```

```
MAE: 430.1505553385447
MSE: 549222.053000202
RMSE: 741.0951713512928
```

# Support Vector Resgressor

```
[ ]  reg_svr = SVR(kernel = 'linear')
     reg_svr.fit(X_train, y_train)
     preds_svr = reg_svr.predict(X_test)
     reg_svr.score(X_test, y_test)
```

```
0.9675832372528995
```

```
[ ]  print('MAE:', metrics.mean_absolute_error(y_true=y_test, y_pred=preds_svr))
     print('MSE:', metrics.mean_squared_error(y_true=y_test, y_pred=preds_svr))
     print('RMSE:', np.sqrt(metrics.mean_squared_error(y_true=y_test, y_pred=preds_svr)))
```

```
MAE: 194.0502450508438
MSE: 153164.08112838
RMSE: 391.3618289107664
```

# Support Vector Resgressor - Tuning

```
param_grid = {'C': [0.1,1], 'gamma': [1,0.1],'kernel': ['linear']}
grid = GridSearchCV(
    reg_svr,
    param_grid,
    refit=True,
    verbose=2)
grid.fit(X_train_sc,y_train)
```

```
grid.best_estimator_
```

```
SVR(C=1, gamma=1, kernel='linear')
```

```
grid.best_score_
```

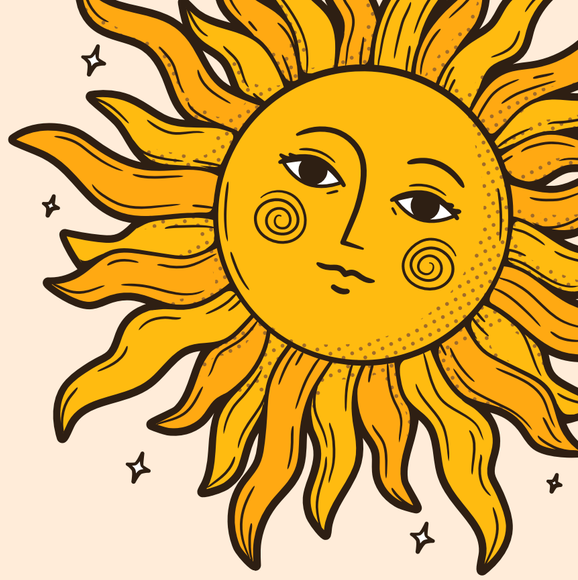
```
0.9565314103894614
```

```
grid_predictions = grid.predict(X_test_sc)
```

```
print('MAE:', metrics.mean_absolute_error(y_true=y_test, y_pred=grid_predictions))
print('MSE:', metrics.mean_squared_error(y_true=y_test, y_pred=grid_predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_true=y_test, y_pred=grid_predictions)))
```

```
MAE: 225.68936225317952
MSE: 180233.46448381466
RMSE: 424.5391200864941
```

# Classification



# Our objective

Predict the

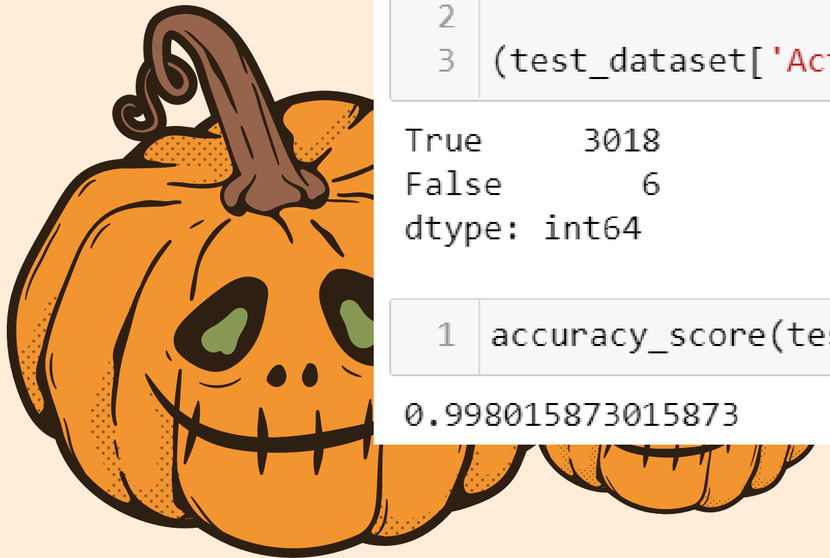
Summary

Profits

# Logistic Regression

```
target = "Summary Profits"

X_train = training.drop(target, axis=1)
y_train = training[target]

X_test = test.drop(target, axis=1)
y_test = test[target]
```

```
classifier = LogisticRegression(random_state = 0)

classifier.fit(X_train, y_train)
```



```
# Counting the records when the actual and the predicitons are the same

(test_dataset['Actual_profit'] == test_dataset['Predict_profit']).value_counts()
```

```
True     3018
False       6
dtype: int64
```

```
accuracy_score(test_dataset['Actual_profit'],test_dataset['Predict_profit'],normalize=True)
```

```
0.998015873015873
```

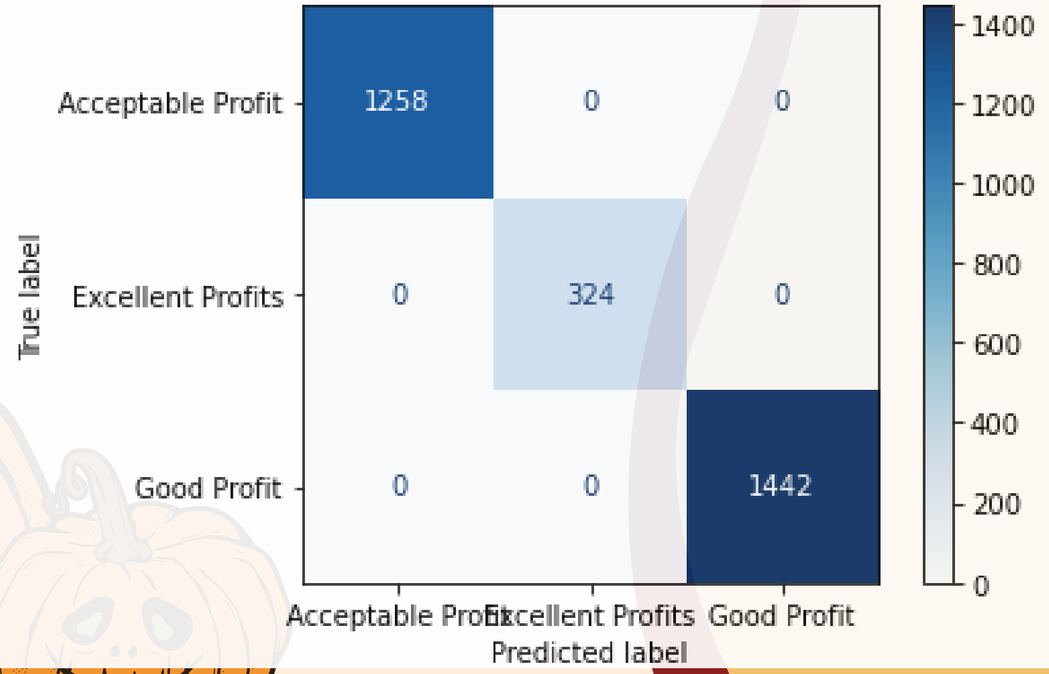
# Confusion matrix

```
# Note: This is a link for more details on the confusion matrix in scikit-learn

cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[1258,    0,    0],
       [   0,  324,    6],
       [   0,    0, 1436]], dtype=int64)
```

```
plot_confusion_matrix(classifier, X=X_test, y_true=y_pred, cmap='Blues')
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x246d3fefd60>
```




# Logistic regression - Tuning

```
# Instantiate Standard Scaler
scaler = StandardScaler()
# Fit & transform data.
scaled_df = scaler.fit_transform(X_train)
```

```
# Instantiate & fit data using PCA
pca = PCA()
pca.fit(scaled_df)
```

```
PCA()
```

```
pipe = make_pipeline(
    PCA(n_components= 9),
    LogisticRegression()
)

pipe.fit(X_train, y_train)
pipe.score(X_test, y_test)
```

```
0.8578042328042328
```

# Classification report & Confusion matrix

```
pred = pipe.predict(X_test)
print(classification_report(y_test, pred))
```

```
                    precision    recall  f1-score   support

Acceptable Profit       0.84      1.00      0.91      1258
Excellent Profits       0.77      0.59      0.67       330
      Good Profit       0.89      0.80      0.84      1436

         accuracy                           0.86      3024
        macro avg       0.83      0.79      0.81      3024
     weighted avg       0.86      0.86      0.85      3024
```

```
plot_confusion_matrix(pipe, X_test, y_test)
plt.grid(False);
```


# Decision Tree

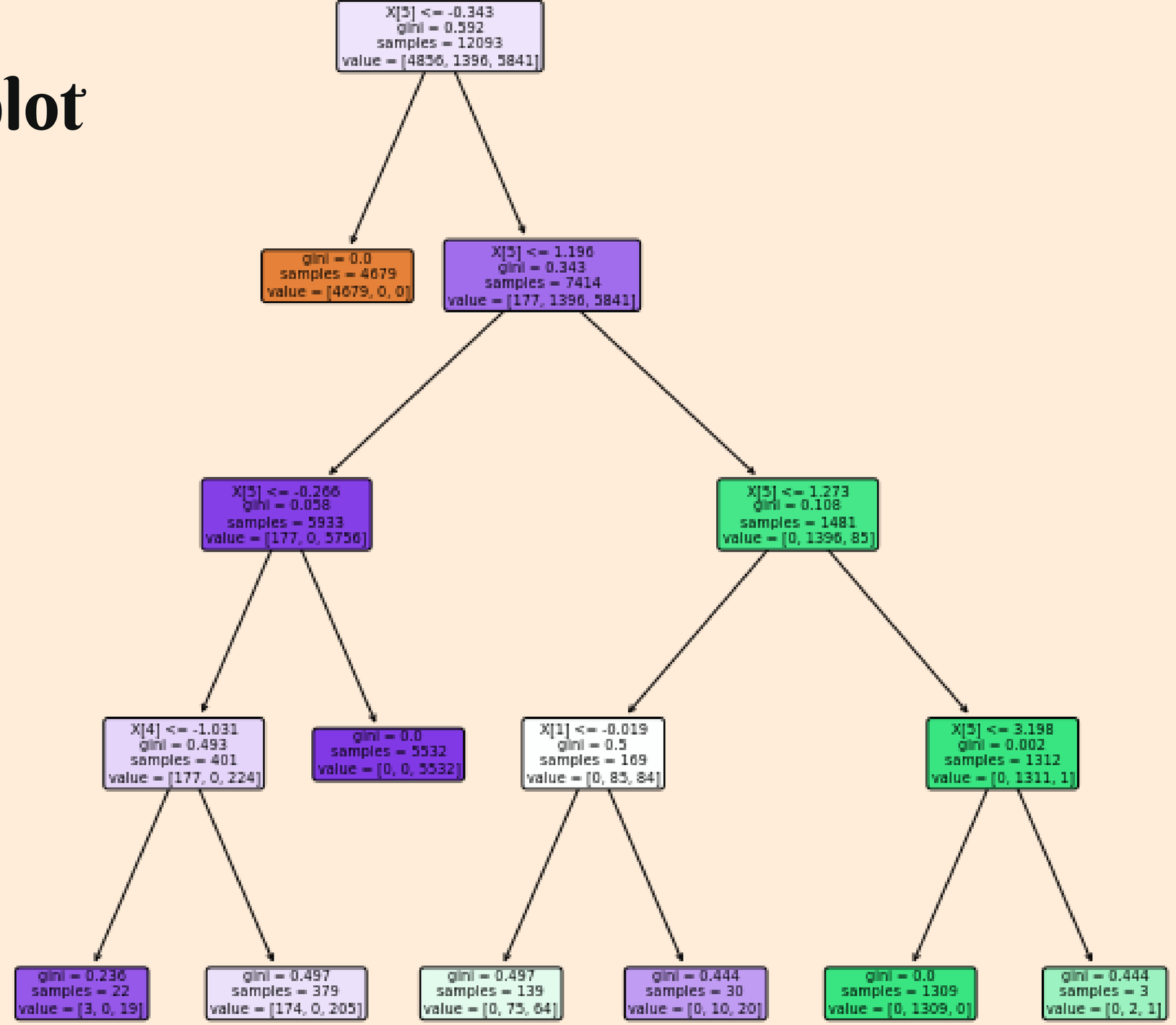
```
scaler = StandardScaler()
# Fit & transform data.
X_train_sc = scaler.fit_transform(X_train)
X_test_sc = scaler.transform(X_test)
```

```
class_tree = DecisionTreeClassifier(criterion='gini', max_depth=4)
class_tree.fit(X_train_sc, y_train)
preds_class = class_tree.predict(X_test_sc)
```

```
val_train = round(class_tree.score(X_train_sc, y_train),2)*100
val_test = round(class_tree.score(X_test_sc, y_test),2)*100

print(f'Training Accuracy: {val_train}%')
print(f'Test Set Accuracy: {val_test}%')
```

```
Training Accuracy: 98.0%
Test Set Accuracy: 98.0%
```

# Decision Tree plot



# Decision tree - Tuning

```
# Classification
param_grid = {
    "criterion": ["gini", "entropy"],
    "max_depth": [2,4,6]
}
grid = GridSearchCV(
    class_tree,
    param_grid,
    cv = 5,
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train, y_train)
```

```
grid.best_score_
```

```
0.9783343608344609
```

```
# Note: we only choose one possibility --> the one happens first
grid.best_params_
```

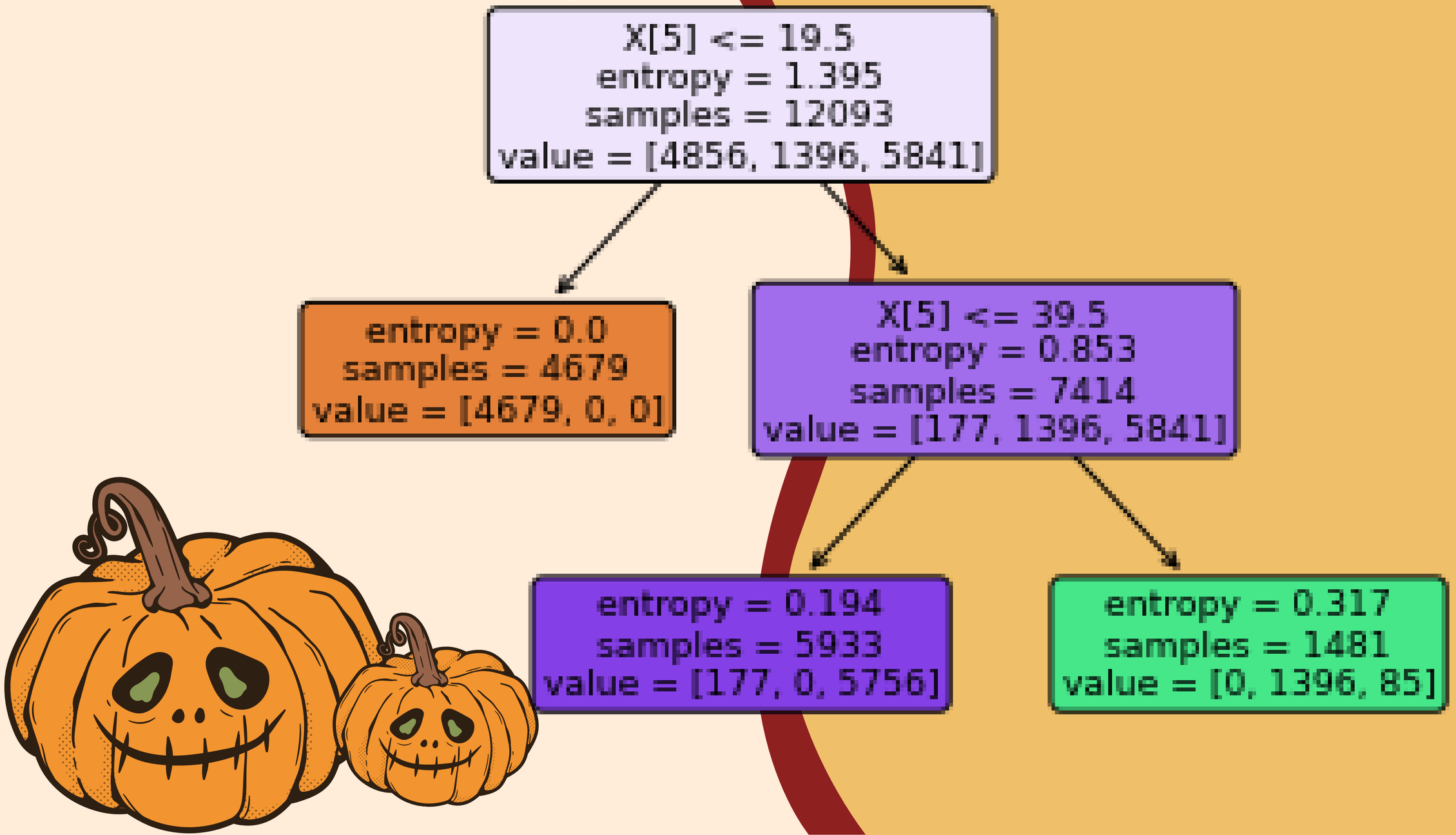
```
{'criterion': 'entropy', 'max_depth': 2}
```

```
grid.best_estimator_
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=2)
```

```
tree.plot_tree(
    grid.best_estimator_, # Access each individual estimator
    filled=True,
    rounded=True,
    )
plt.show()
```

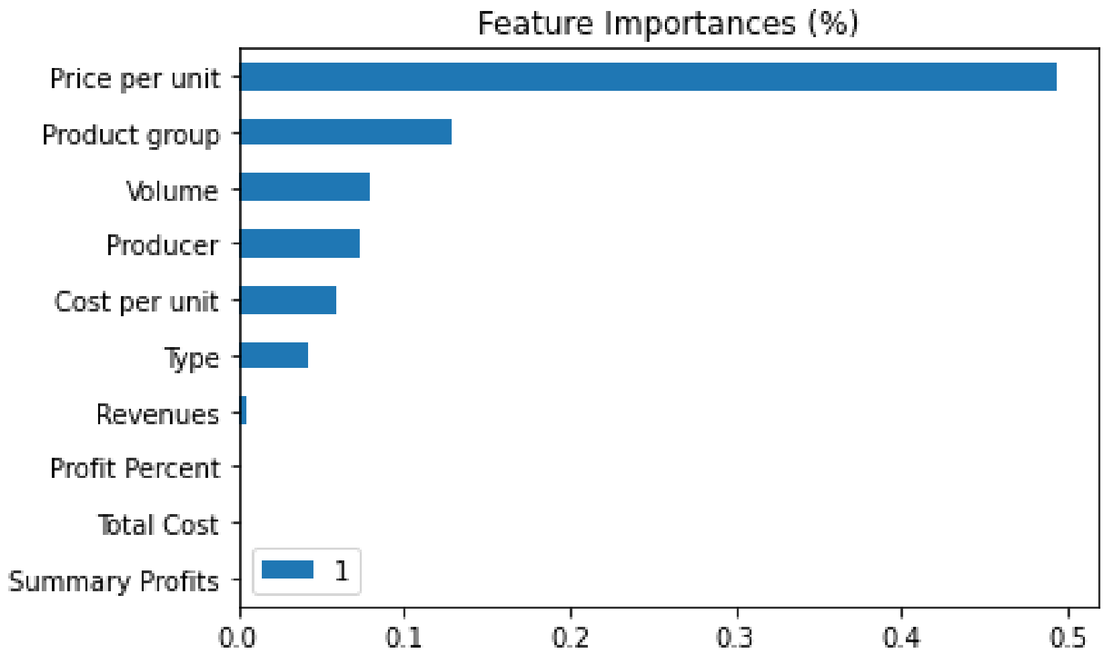
# Descion Tree Plot - Tuning



# Random forest classifier

```
class_forest = RandomForestClassifier(n_estimators = 6, criterion = 'gini', random_state = 0)
class_forest.fit(X_train_sc, y_train)
preds_class = class_forest.predict(X_test_sc)
```

```
pd.DataFrame(dict(zip(df.columns, class_forest.feature_importances_)), index = [1])\
.T\
.sort_values(1, ascending=True)\
.plot(kind="barh", title="Feature Importances (%)");
```

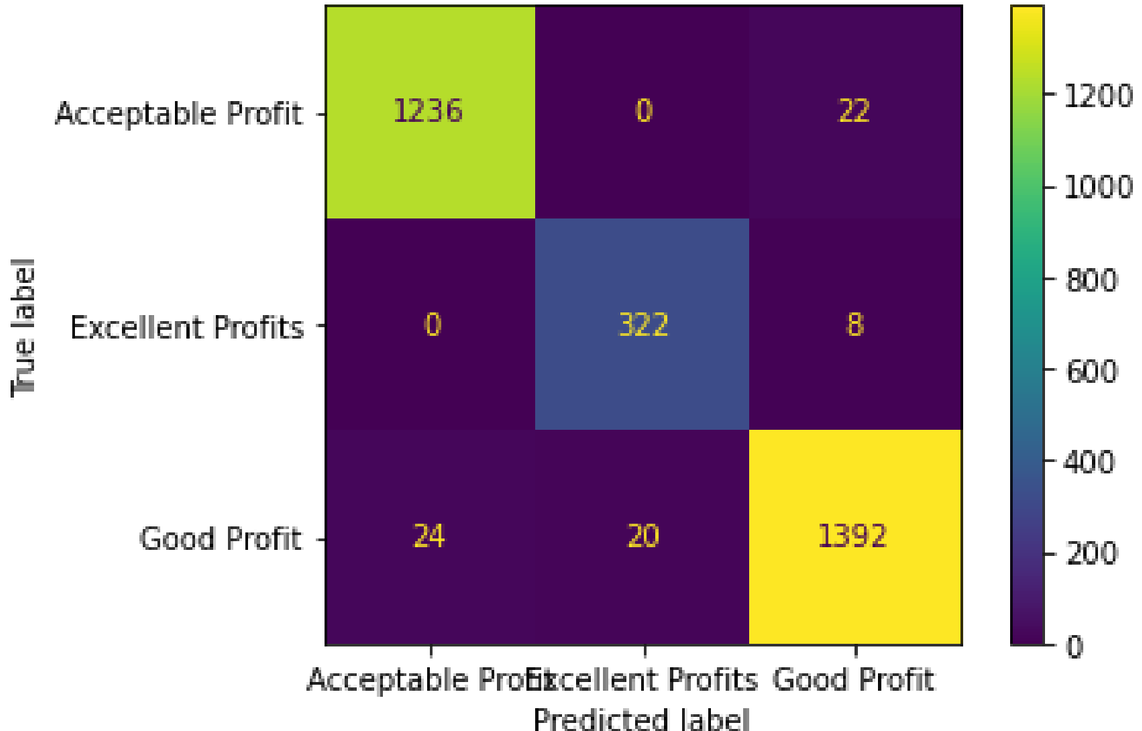



# Accuracy & Confusion matrix

```
val_train = round(class_forest.score(X_train_sc, y_train),2)*100
val_test = round(class_forest.score(X_test_sc, y_test),2)*100

print(f'Training Accuracy: {val_train}%')
print(f'Test Set Accuracy: {val_test}%')
```

```
Training Accuracy: 100.0%
Test Set Accuracy: 98.0%
```

```
plot_confusion_matrix( class_forest,X_test_sc, y_test);
```



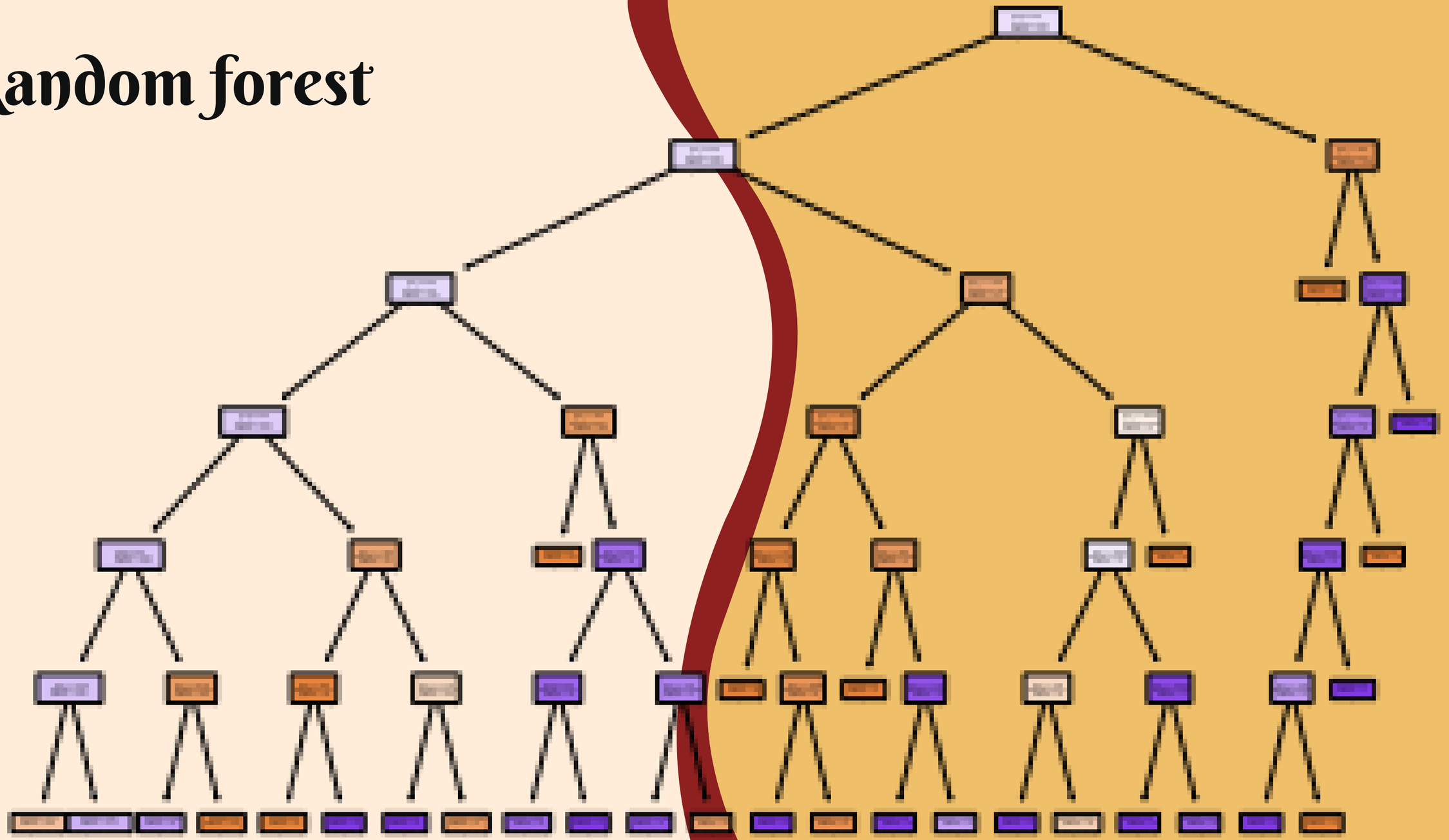

# Random forest - Tuning

```
# Classification
param_grid = {
    "n_estimators": [10,20,30],
    "criterion": ["gini", "entropy"],
    "max_depth": [2,4,6]
}
grid = GridSearchCV(
    class_forest,
    param_grid,
    cv = 5,
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train, y_train)
```

```
grid.best_score_
```

```
0.8997778135665027
```

# Random forest



# Support vector classifier

```
class_sv = SVC(kernel = 'linear', random_state = 0)
class_sv.fit(X_train_sc, y_train)
preds_class = class_sv.predict(X_test_sc)
```

```
val_train = round(class_sv.score(X_train_sc, y_train),2)*100
val_test = round(class_sv.score(X_test_sc, y_test),2)*100
print(f'Training Accuracy: {val_train}%')
print(f'Test Set Accuracy: {val_test}%')
```

```
Training Accuracy: 99.0%
Test Set Accuracy: 99.0%
```

# Support vector classifier - tuning

```
scores = cross_validate(class_sv, X_train_sc, y_train,scoring='accuracy')
scores
```

```
{'fit_time': array([1.31352401, 1.35836458, 1.29852867, 1.24067974, 1.26557517]),
 'score_time': array([0.11768651, 0.13064504, 0.11365962, 0.11971259, 0.11569238]),
 'test_score': array([0.98387764, 0.98759818, 0.98263745, 0.9751861 , 0.97766749])}
```

```
param_grid = {'C': [0.1,1], 'gamma': [1,0.1],'kernel': ['poly', 'sigmoid']}
grid = GridSearchCV(
    class_sv,
    param_grid,
    refit=True,
    verbose=2)
grid.fit(X_train_sc,y_train)
```

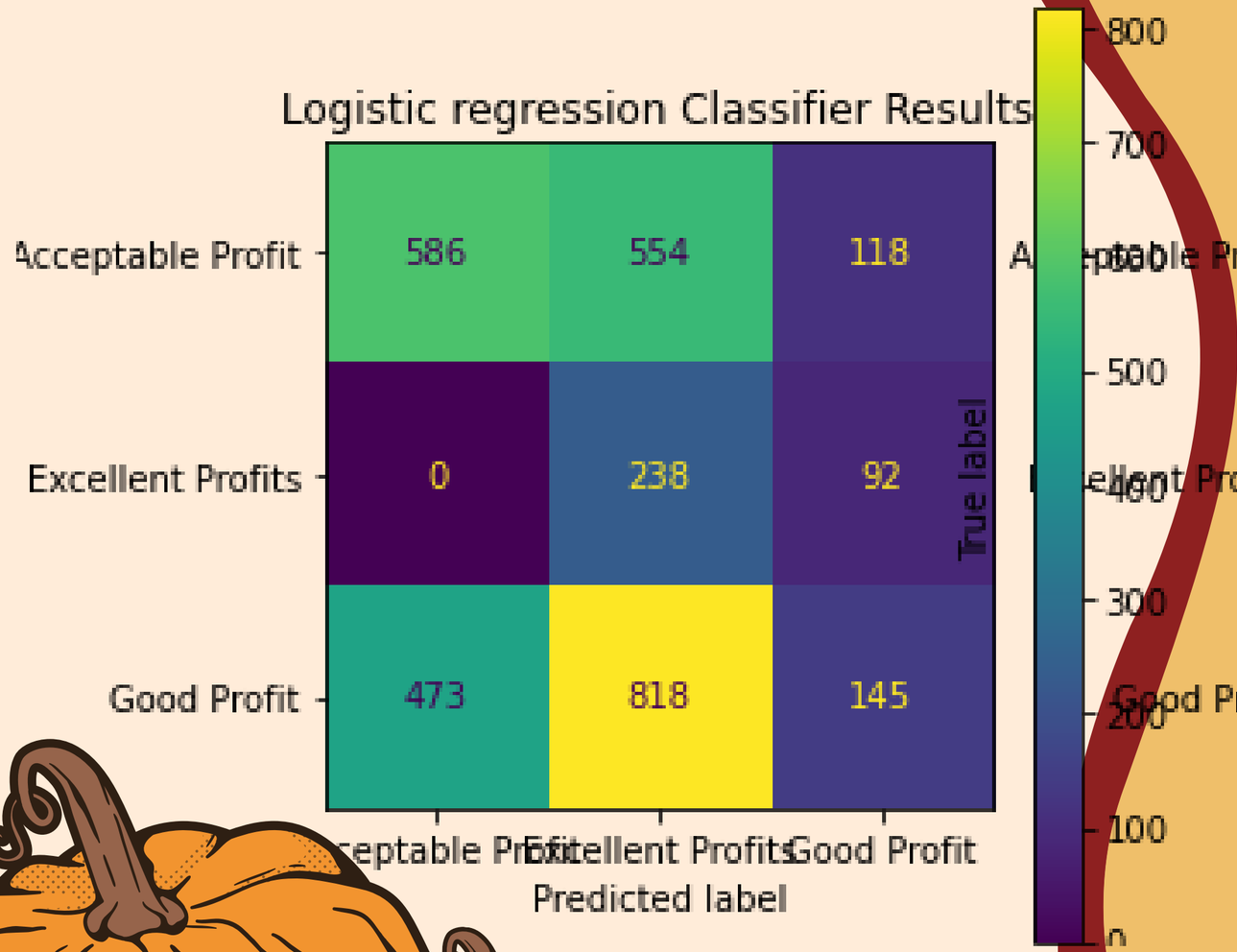
```
grid.best_estimator_
```

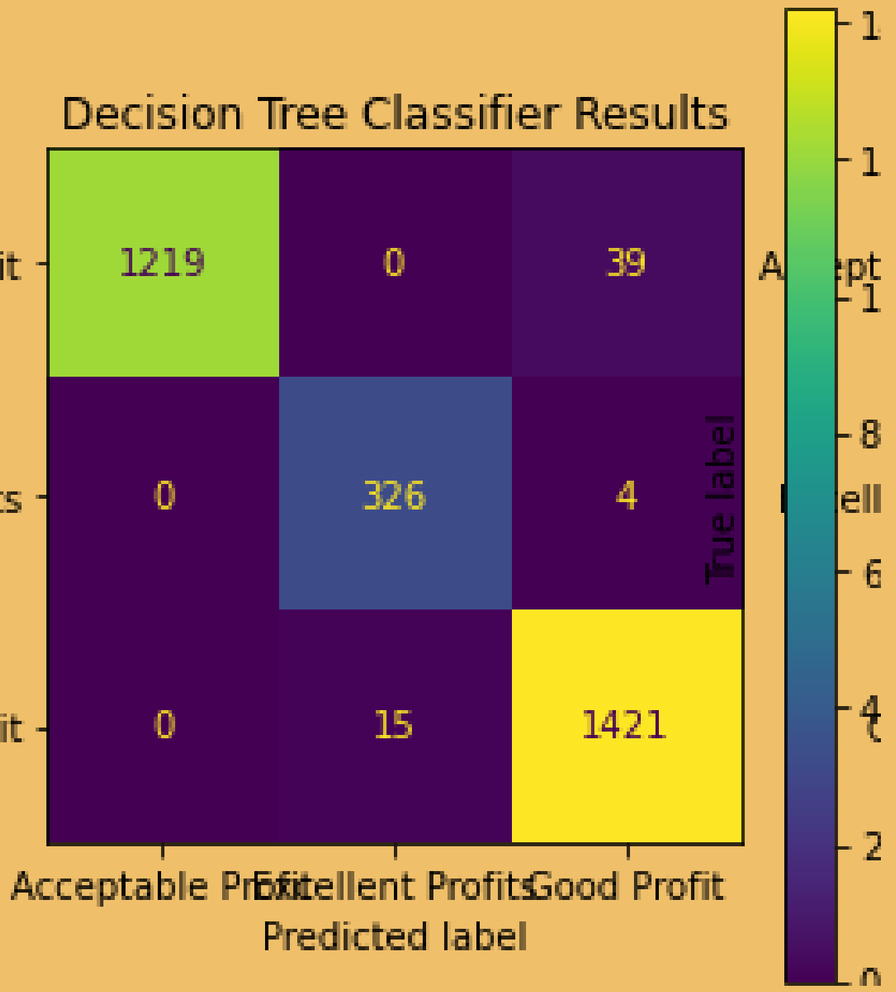
```
SVC(C=1, gamma=1, kernel='poly', random_state=0)
```

```
grid.best_score_
```

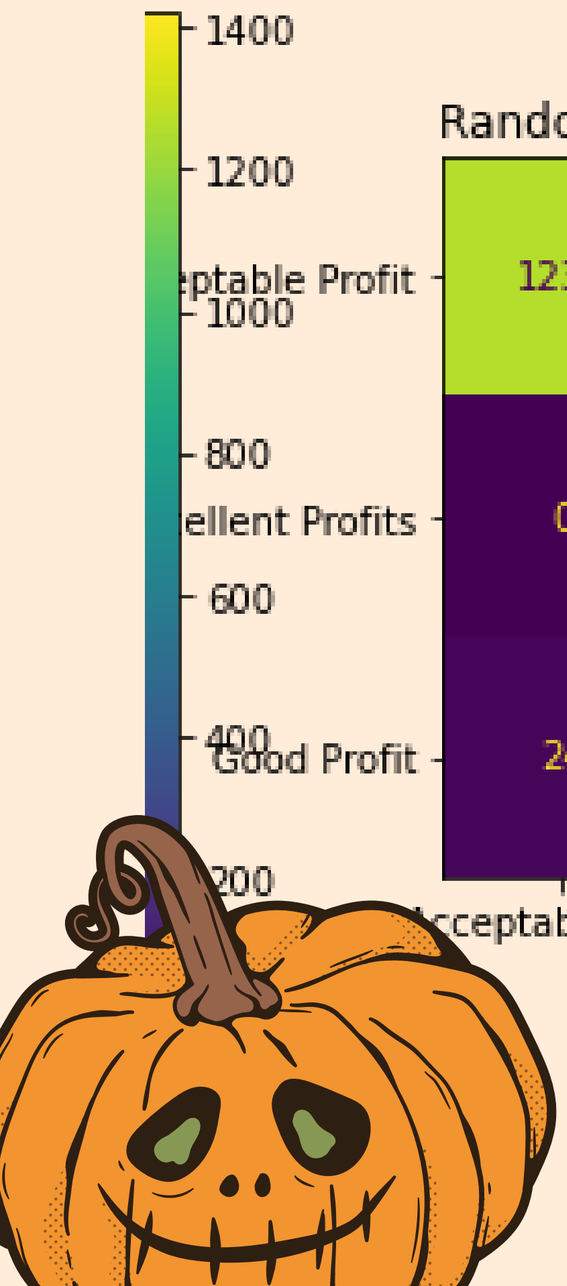
```
0.9692380865432912
```

# all confusion matrix for all plots without tuning

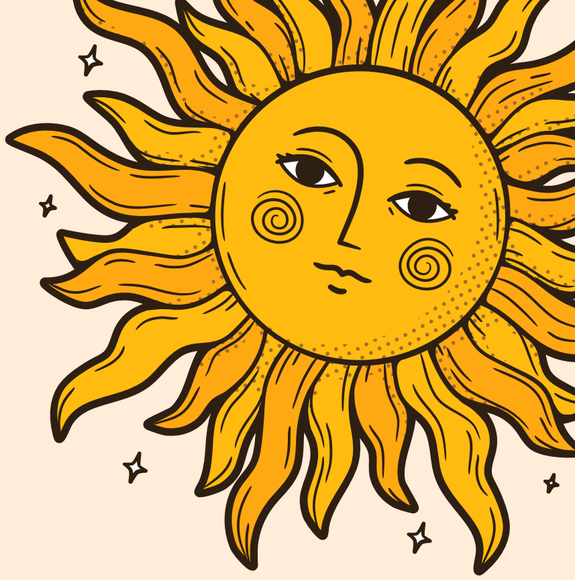



# all confusion matrix for all plots without tuning



# Pipeline



# Classification Pipeline

**The best model is the decision tree**

```
pipe = make_pipeline(
    StandardScaler(),
    DecisionTreeClassifier(criterion='entropy', max_depth=2)
)

pipe.fit(X_train, y_train)
pipe.score(X_test, y_test)
```

```
0.9811507936507936
```

# Regression Pipeline

**The best model is the linear regression model**

In [38]:
```
# Create a Pipeline for our model
pipe = make_pipeline(
    # scale columns
    StandardScaler(),
    # apply the model
    LinearRegression()
)

pipe.fit(X_train,y_train)
pipe.score(X_test, y_test)
```

Out[38]: 0.9785664818077269

# Thank you for listening!