SWISS BANKNOTE CONTERFEIT DETECTION

Presented by Team AWS

Our Team

Name

Role: Presenter 1

Name

Role: Presenter 2

Project Lead
Name:
Assistant Project Lead
Name: Jimoh Abdulsomad Abiola
Query Analysts
Name: Temitope Flourish Oke

Active Members

Name	Role
	Team Lead
Jimoh Abdulsomad Abiola	Assitant Team Lead
Temitope Flourish Oke	Query Ananlyst
Somya Kumari	Modelling and Evaluation
Komal	Powerpoint Presentation
Faith Wambugu	Processing
Emmanuel Oyetunji	Testing and evaluation

Problem Statement

HAMOYE

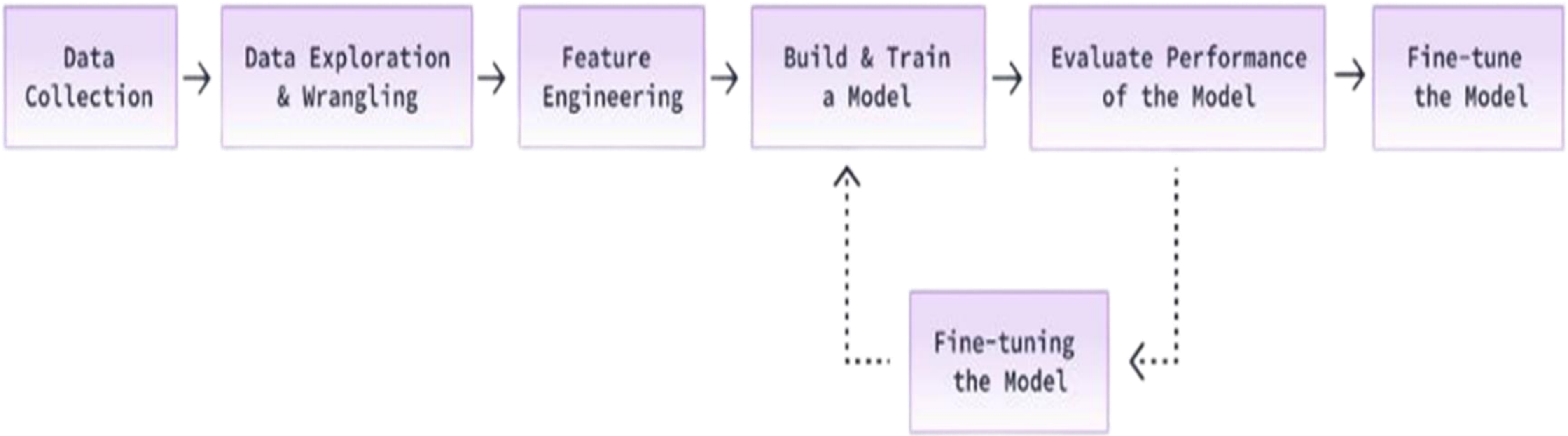
- One of a nation's most valuable assets is its banknotes
- To cause differences in the amount of money in the financial market, some criminals introduce fake notes that look like the actual notes
- Humans find it challenging to distinguish between real and fake banknotes, in part because they have many characteristics
- As fake notes are meticulously made, an effective algorithm that can predict whether a banknote is real or not is necessary

Existing Solutions

- Swiss banknote counterfeit detection using naive-bayes-classifier
- Swiss banknote counterfeit detection using svm-classifier
- Swiss banknote counterfeit detection using neural-networks
- Swiss banknote counterfeit detection using logistic-regression
- Swiss banknote counterfeit detection using decision-tree-classifier
- Swiss banknote counterfeit detection using k-means-clustering
- Swiss banknote counterfeit detection using random-forest-classifier

Our Approach

Flow Process



Dataset Description

- We used near-perfect data for this problem sourced from Kaggle
- The dataset is available here
- The cleaned data from the exploratory data analysis (EDA) is used to design the machine learning model
- The dataset includes information about the shape of the bill, as well as the label
- It is made up of 200 banknotes in total, 100 for genuine/counterfeit each
- Attributes:
- ➢ counterfeit: Whether a banknote is counterfeit (1) or genuine (0)
- ➢ Length: Length of bill (mm)
- ➢ Left: Width of left edge (mm)
- ➢ Right: Width of right edge (mm)
- ➢ Bottom: Bottom margin width (mm)
- ➢ Top: Top margin width (mm)
- ➢ Diagonal: Length of diagonal (mm)

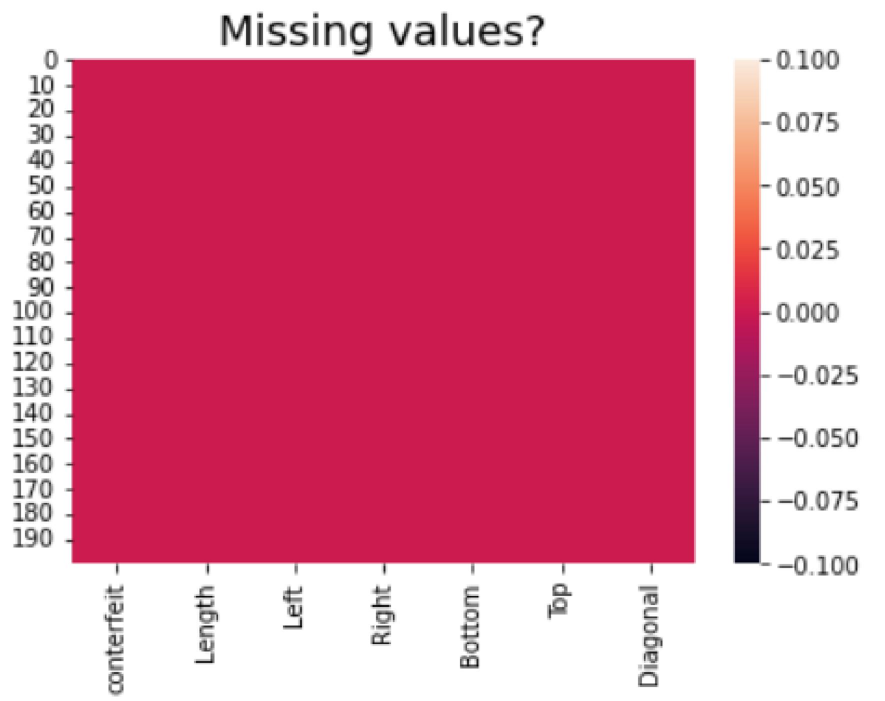
	conterfeit	Length	Left	Right	Bottom	Top	Diagonal
0	0	214.8	131.0	131.1	9.0	9.7	141.0
1	0	214.6	129.7	129.7	8.1	9.5	141.7
2	0	214.8	129.7	129.7	8.7	9.6	142.2
3	0	214.8	129.7	129.6	7.5	10.4	142.0
4	0	215.0	129.6	129.7	10.4	7.7	141.8

Data Exploration

```
bnote.describe()
```

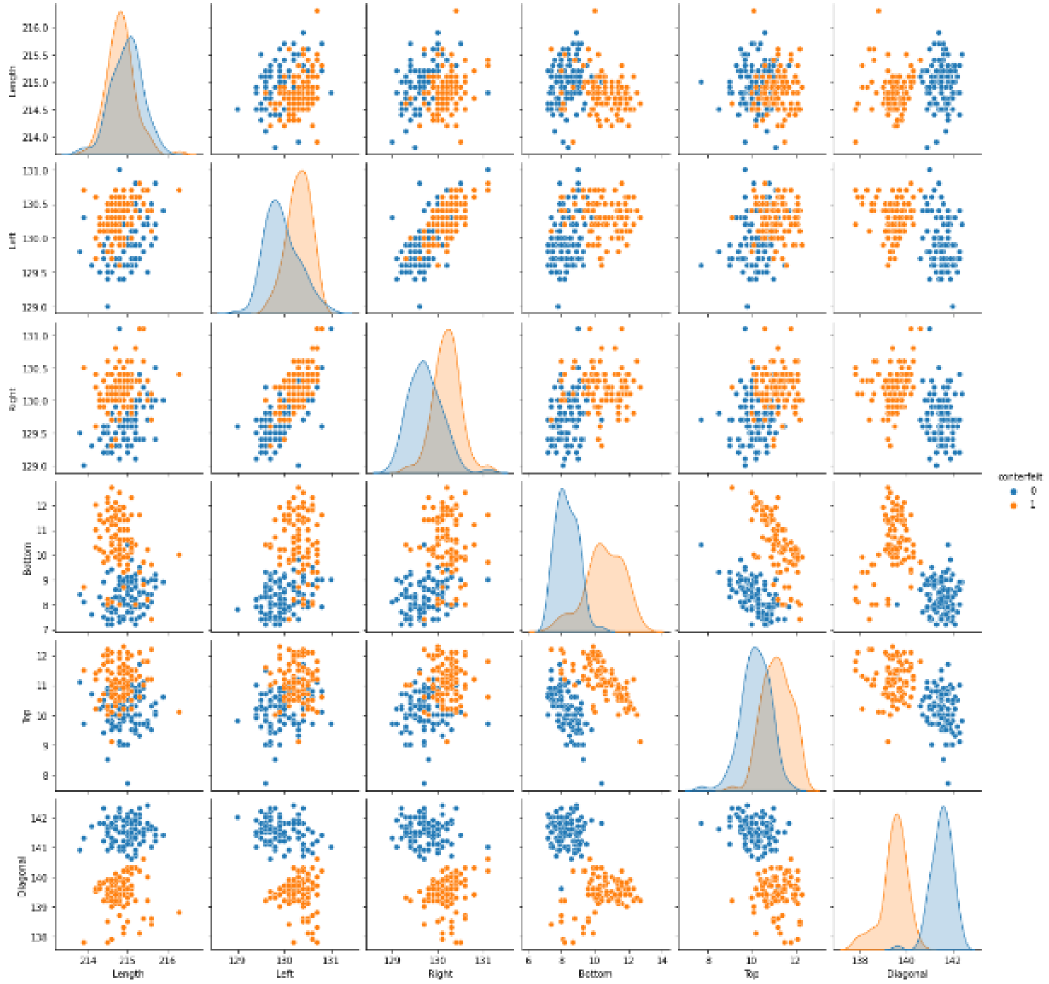
	conterfeit	Length	Left	Right	Bottom	Top	Diagonal
count	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000
mean	0.500000	214.896000	130.121500	129.956500	9.417500	10.650500	140.483500
std	0.501255	0.376554	0.361026	0.404072	1.444603	0.802947	1.152266
min	0.000000	213.800000	129.000000	129.000000	7.200000	7.700000	137.800000
25%	0.000000	214.600000	129.900000	129.700000	8.200000	10.100000	139.500000
50%	0.500000	214.900000	130.200000	130.000000	9.100000	10.600000	140.450000
75%	1.000000	215.100000	130.400000	130.225000	10.600000	11.200000	141.500000
max	1.000000	216.300000	131.000000	131.100000	12.700000	12.300000	142.400000

```
sns.heatmap(bnote.isnull())
plt.title("Missing values?", fontsize = 18)
plt.show()
```

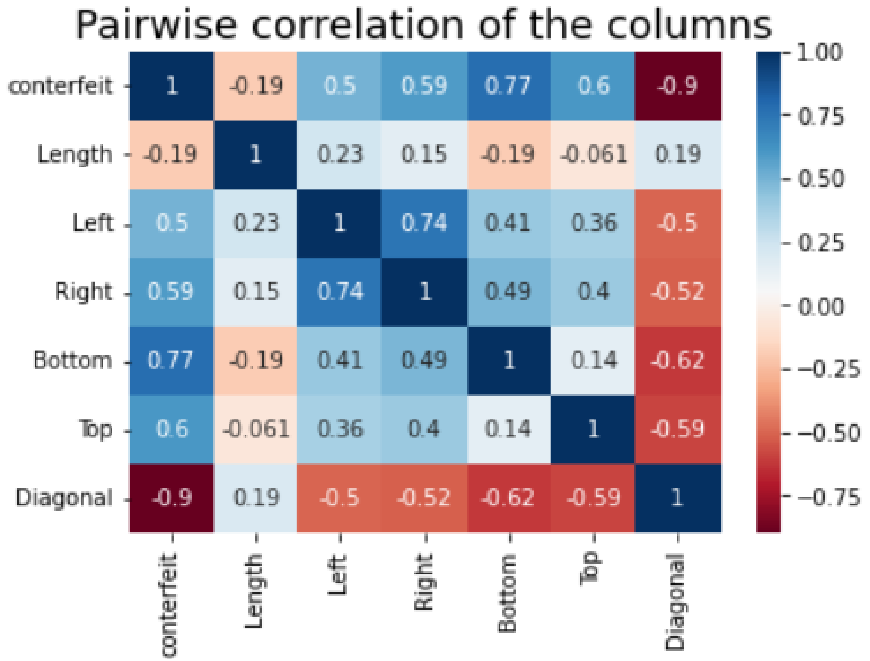


Data Wrangling

```
# Pairwise relationships depending on counterfeit
sns.pairplot(bnote, hue = "conterfeit")
plt.show()
```

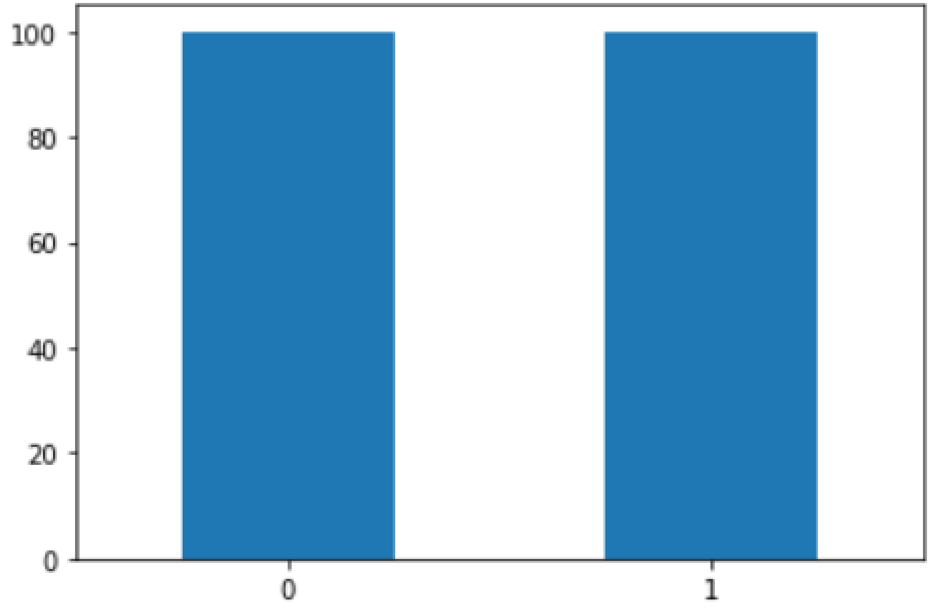



```
sns.heatmap(bnote.corr(), annot = True, cmap="RdBu")
plt.title("Pairwise correlation of the columns", fontsize = 18)
plt.show()
```



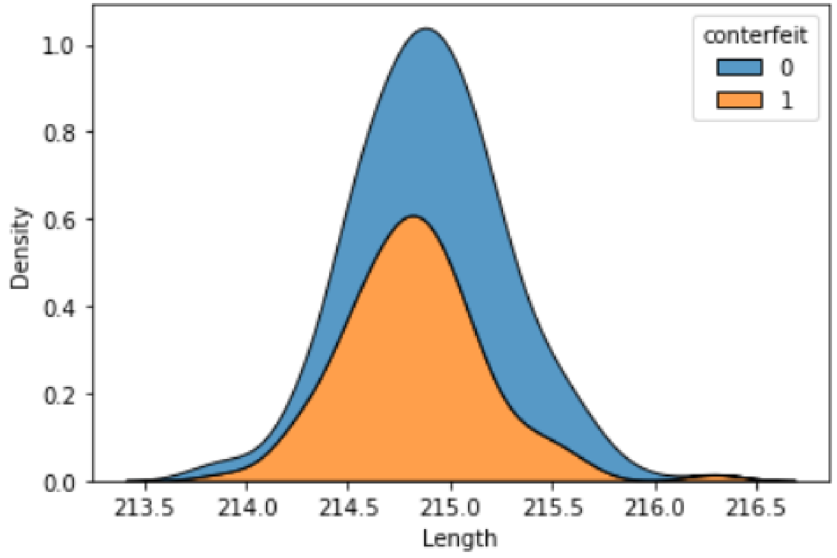
Data Wrangling

```
bnote["conterfeit"].value_counts().plot(kind="bar")
plt.xticks(rotation='horizontal')
plt.show()
```



The data is fairly balanced

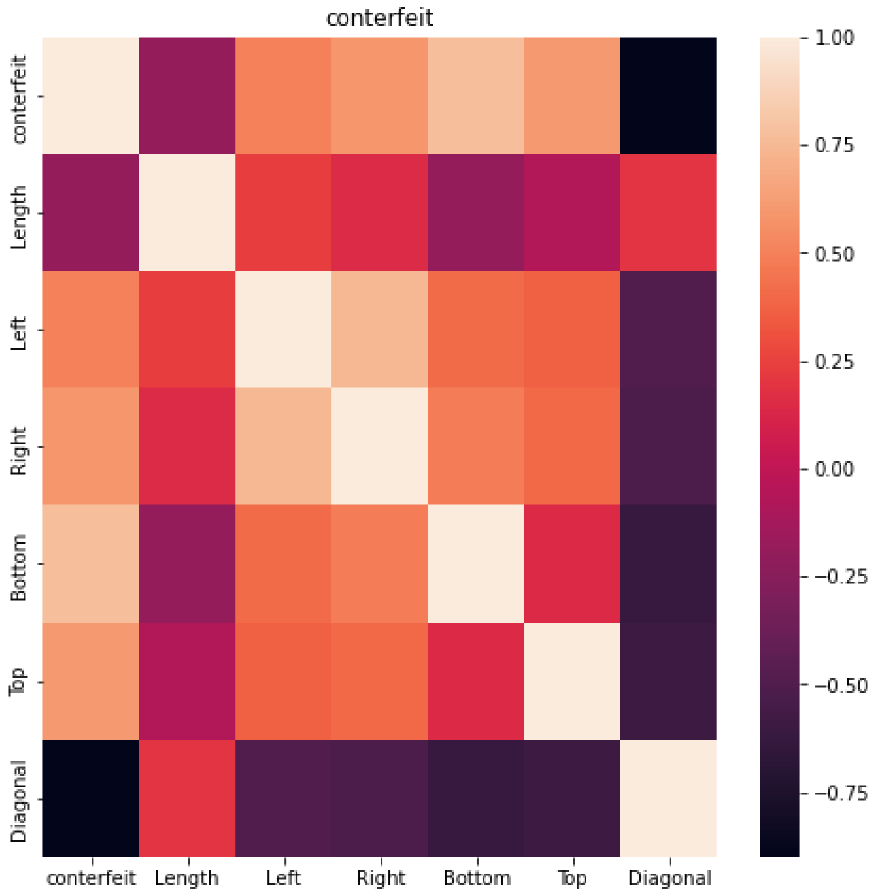
```
sns.kdeplot(data=bnote, x ='Length', hue='conterfeit', multiple='stack')
plt.show()
```



We have more of conterfeit compared to length

Feature Engineering

```
plt.figure(figsize=(8, 8))
num_features = new.select_dtypes(['int', 'float'])
corr_mat = num_features.corr()
sns.heatmap(data=corr_mat)
plt.title('conterfeit')
plt.show()
```



Heatmap of numerical feature in the training dataset

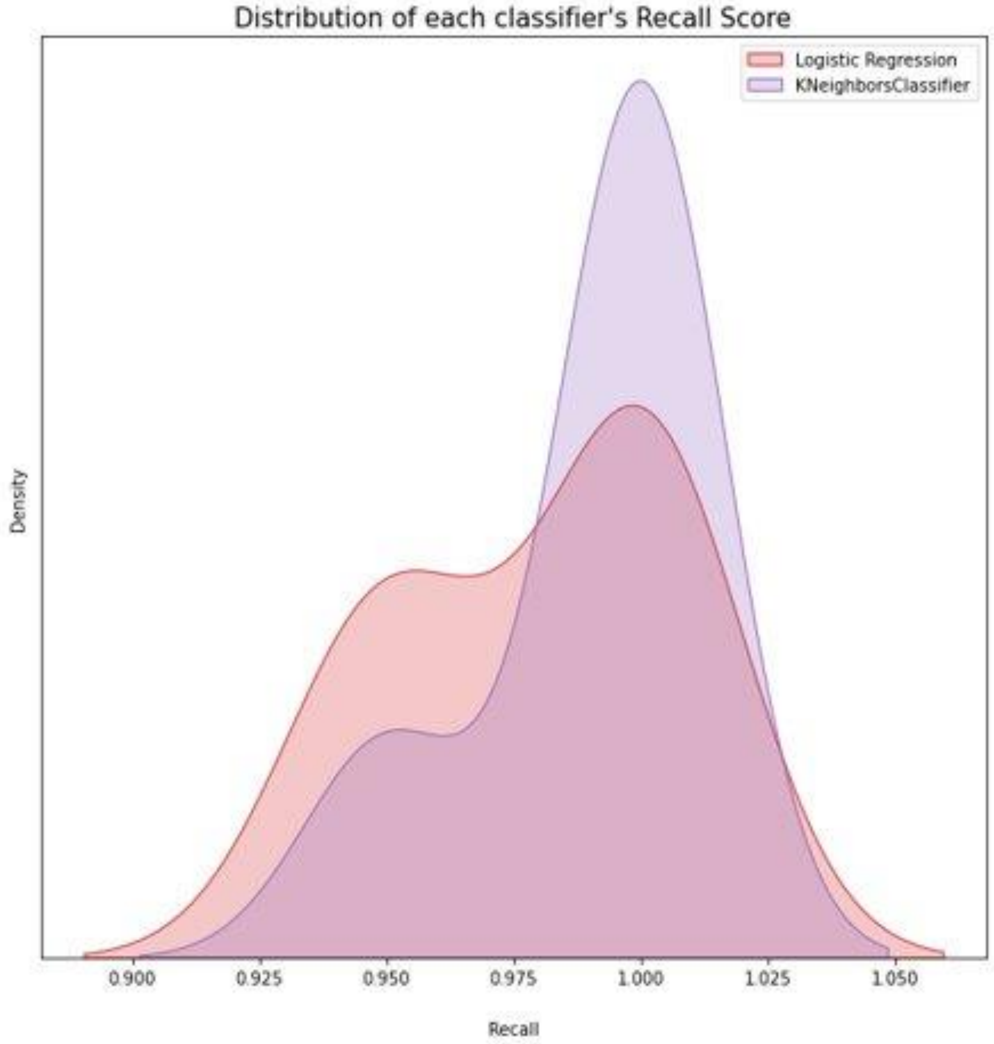
Build & Train the Model

```
bnote = bnote.reindex(np.random.permutation(bnote.index))

X = bnote.drop(columns = "conterfeit")
y = bnote["conterfeit"]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

from sklearn.preprocessing import StandardScaler
st = StandardScaler()
X_train = st.fit_transform(X_train)
```

Models Test, Train and ROU-AUC Score

```
# creating a seed
fixed_seed = 42

# Models for training
models = [LogisticRegression(random_state=fixed_seed), KNeighborsClassifier(), RandomForestClassifier(random_state=fixed_seed),
          DecisionTreeClassifier(random_state=fixed_seed),  XGBC(seed=fixed_seed)]

test_score = []
train_score = []
roc_auc_lst = []

# Loops through each mosel and get their score
for model in models:
    model.fit(X_train, y_train)
    proba = model.predict_proba(X_test)[:, 1]
    test = model.score(X_test, y_test)
    train = model.score(X_train, y_train)
    roc_auc = roc_auc_score(y_test, proba)
    test_score.append(test)
    train_score.append(train)
    roc_auc_lst.append(roc_auc)
```

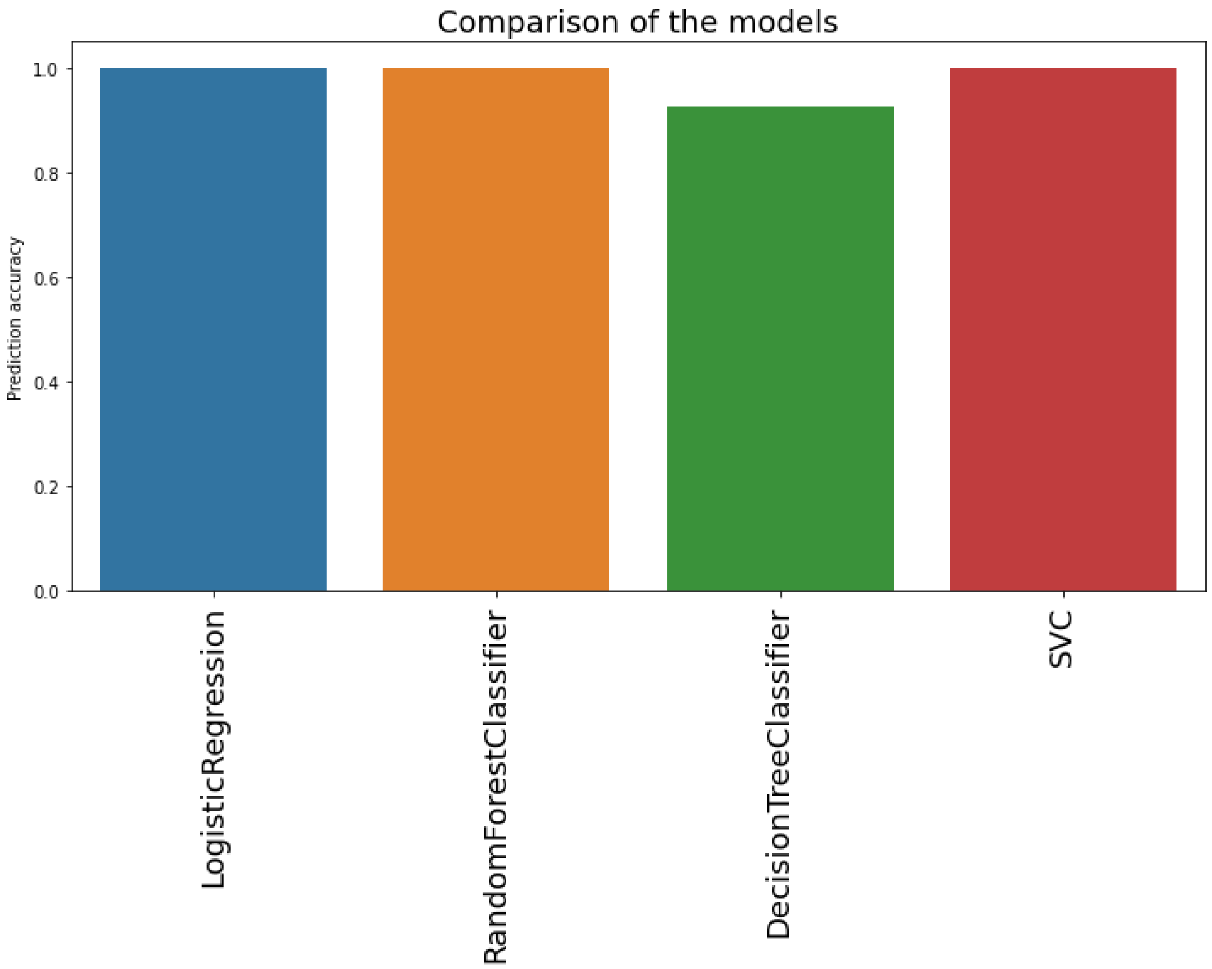
Build & Train the Model



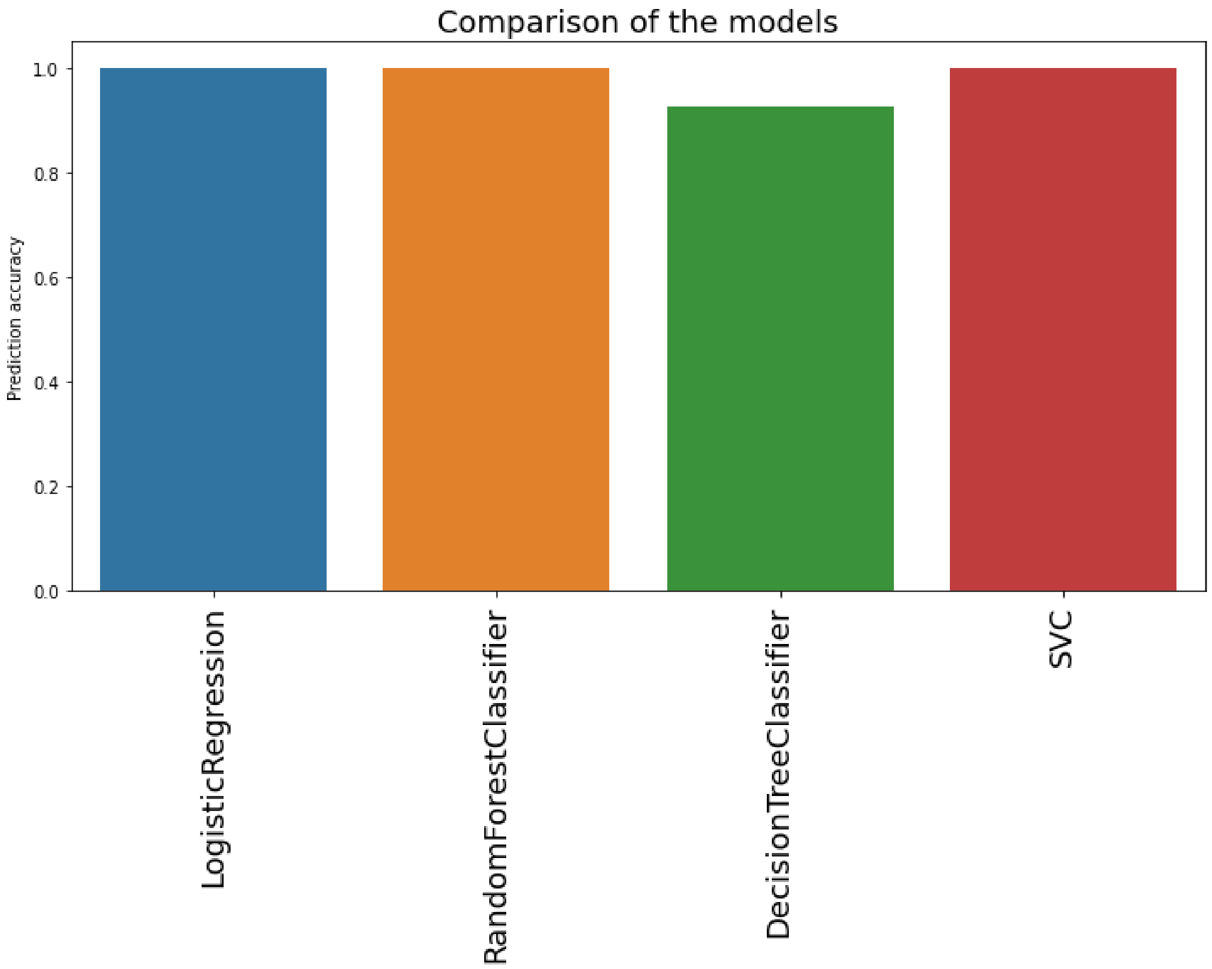

```
models_score = {
    'Train Score': train_score,
    'Test_Score': test_score,
    "ROC_AUC Score": roc_auc_lst
}

pd.DataFrame(models_score, index=['LogisticRegression','KNeighborsClassifier', 'RandomForestClassifier',
                                  'DecisionTreeClassifier', 'XGBC'])
```

	Train Score	Test_Score	ROC_AUC Score
LogisticRegression	0.99375	0.45	1.0
KNeighborsClassifier	0.98750	0.45	0.5
RandomForestClassifier	1.00000	0.45	0.5
DecisionTreeClassifier	1.00000	0.45	0.5
XGBC	0.99375	0.45	0.5

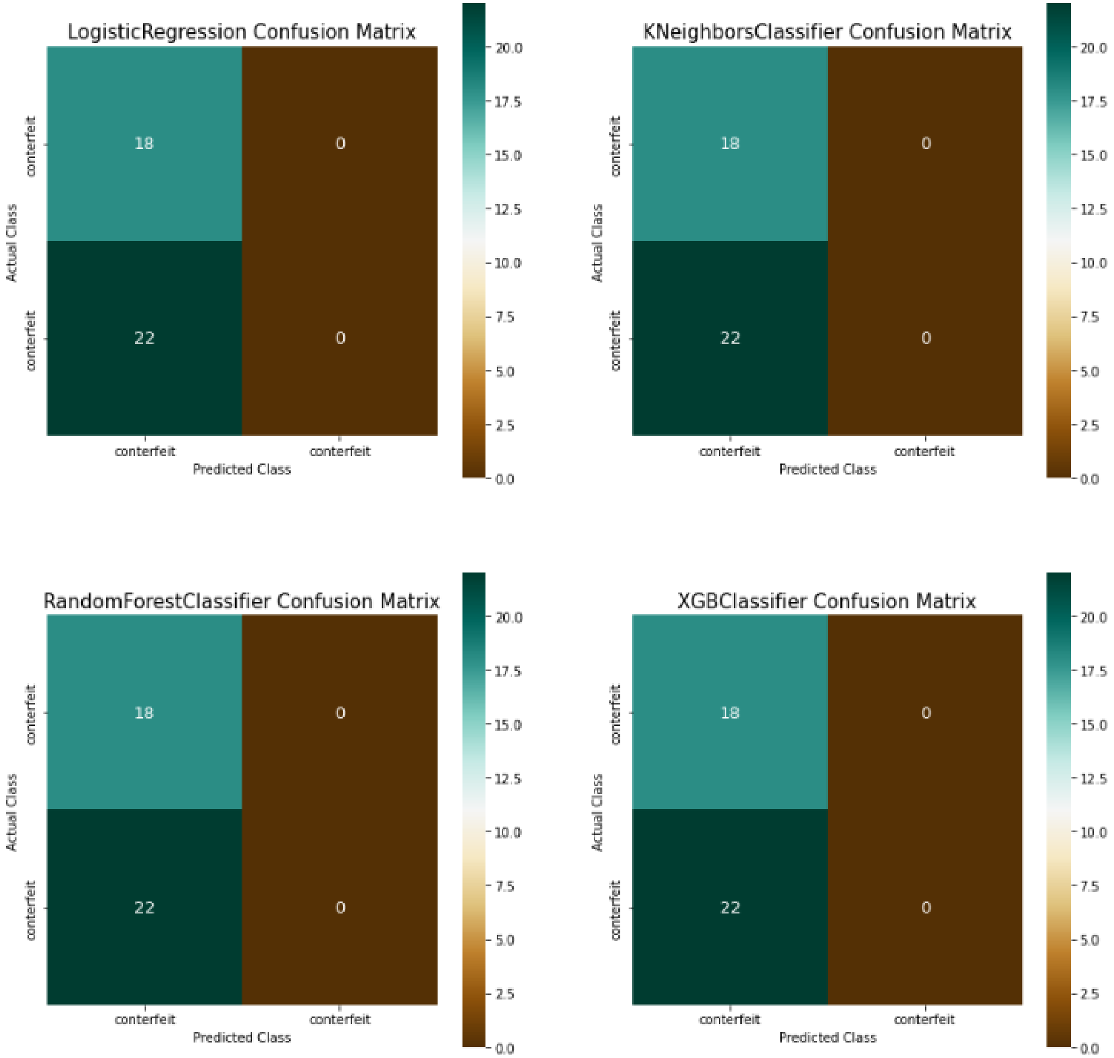
Build & Train the Model

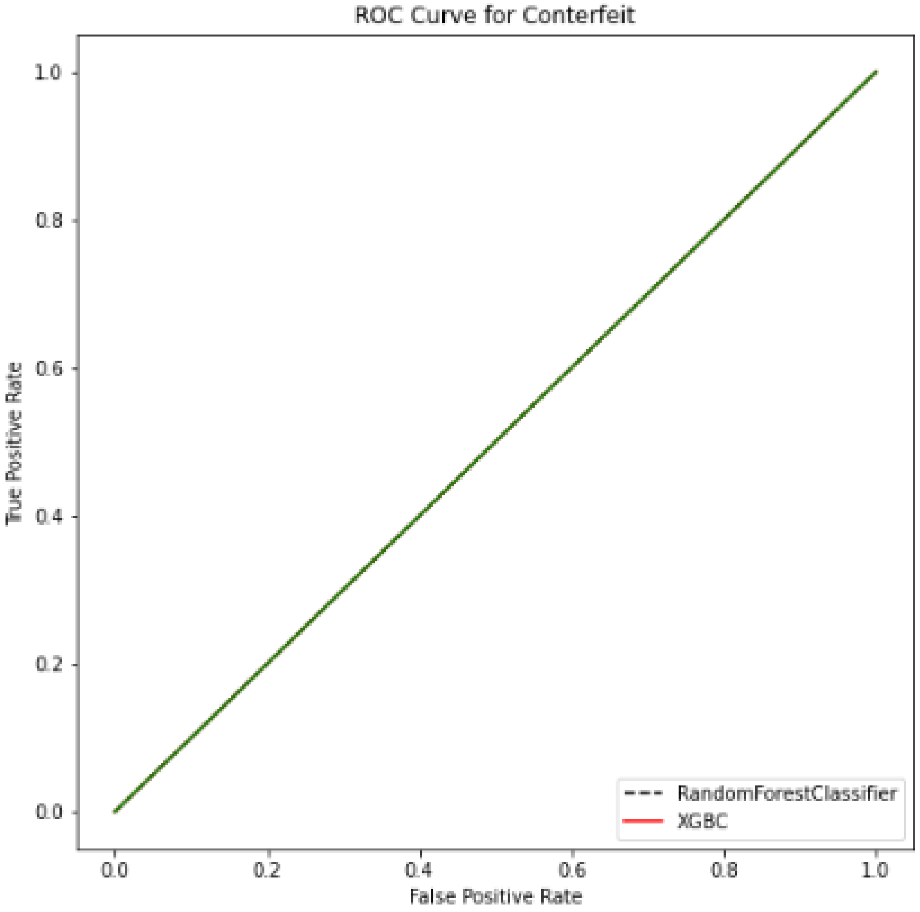


Build & Train the Model



Build & Train the Model





Above we can see the confusion matri for each models

Evaluate Performance of the Model

```
# Function to calculate model average error and accuracy
def evaluate(model, test_features, test_labels):
    predictions = model.predict(test_features)
    errors = abs(predictions - test_labels)
    print('Model Performance')
    print('='*30)
    avg_error = 'Average Error: {:0.4f} degrees.'.format(np.mean(errors))
    print(avg_error)
    accuracy = accuracy_score(y_test, predictions)
    print('Accuracy: {:0.2f}%.'.format(100 * accuracy))

    return [np.mean(errors), accuracy]

# Function to show model performance difference
def improvement(new_score, base):
    print('Improvement Error: {:0.2f}%.'.format( 100 * (new_score[0] - base[0]) / base[0]))
    print('Improvement Accuracy: {:0.2f}%.'.format( 100 * (new_score[1] - base[1]) / base[1]))
    print('=' * 30)
```

- This function was created to evaluate the performance of the model

Fine-tune the Model

Performance of Best Random Search Model

```
best_random = rfc_random.best_estimator_
random_performance = evaluate(best_random, X_test, y_test)
improvement(random_performance, base_performance)
```

```
Model Performance
==============================
Average Error: 0.5500 degrees.
Accuracy: 45.00%.
Improvement Error: 0.00%.
Improvement Accuracy: 0.00%.
==============================
```

Hyper Parameter Tuning 1

```
best_grid = grid_search.best_estimator_
grid_performance = evaluate(best_grid, X_test, y_test)
improvement(grid_performance, base_performance)
```

```
Model Performance
==============================
Average Error: 0.5500 degrees.
Accuracy: 45.00%.
Improvement Error: 0.00%.
Improvement Accuracy: 0.00%.
==============================
```

Hyper Parameter Tuning 2

```
Model Performance
==============================
Average Error: 0.5500 degrees.
Accuracy: 45.00%.
Improvement Error: 0.00%.
Improvement Accuracy: 0.00%.
==============================
```

Hyper Parameter Tuning 3

- A big decrease in performance, yet the same best parameters was given
- This indicates we have reached diminishing returns for hyperparameter tuning
- We could continue, but the returns would be the same output in terms of parameters

Final Model

Final Model

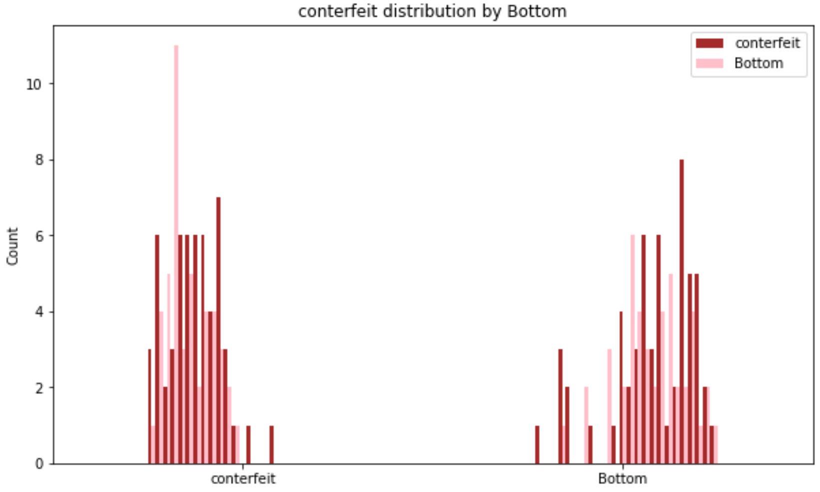
```
final_model = grid_search.best_estimator_
print('Final Model Parameter:')
print('='*30)
print(final_model.get_params())
print('='*30)
grid_final_accuracy = evaluate(final_model, X_test, y_test)
```

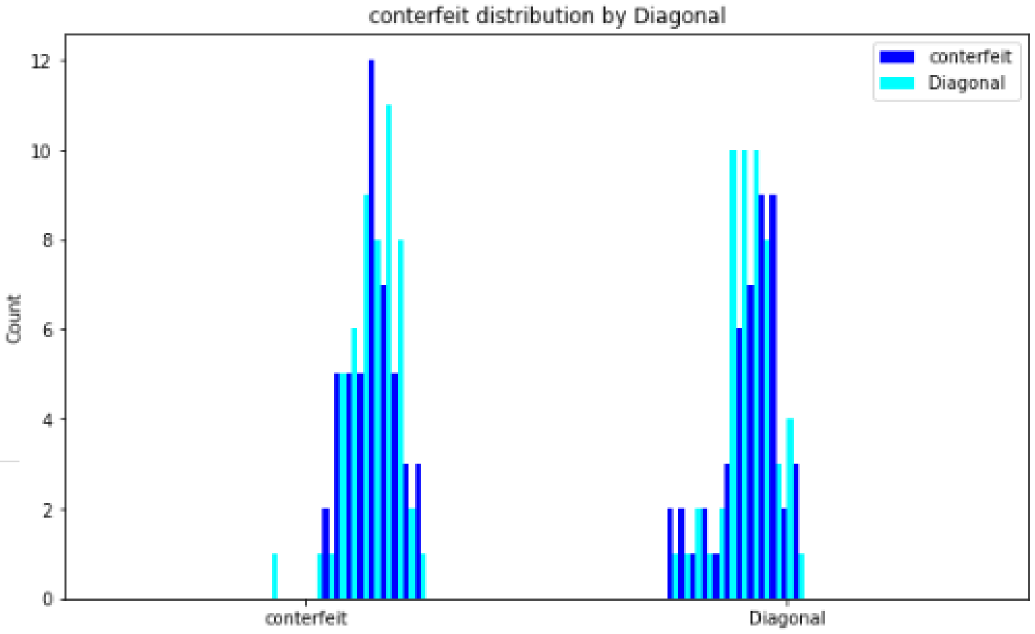
```
Final Model Parameter:
==============================
{'bootstrap': False, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': 40, 'max_features': 'auto', 'max
_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'min_samples_split': 2, 'min_weig
ht_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': 42, 'verbose': 0, 'warm_star
t': False}
==============================
Model Performance
==============================
Average Error: 0.5500 degrees.
Accuracy: 45.00%.
```

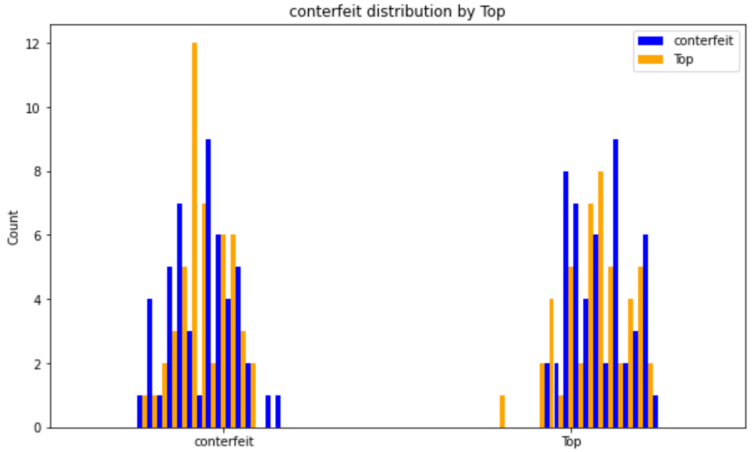
Summary

By using the results, we summarized the data and demonstrated some easy methods of analysis:

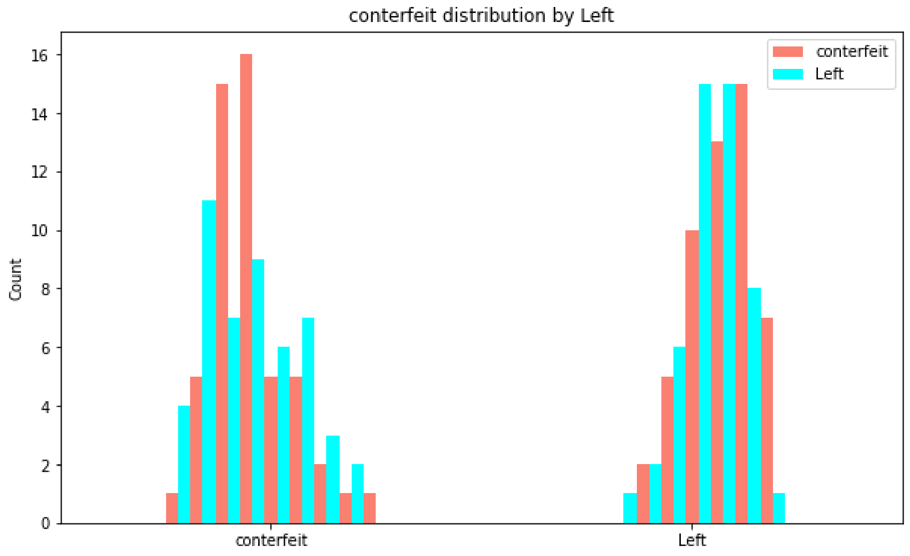
- Simply employing diagonal variables has produced excellent classification results
- Top and Bottom are two excellent variables to include in the predict model in order to categorize counterfeit banknotes

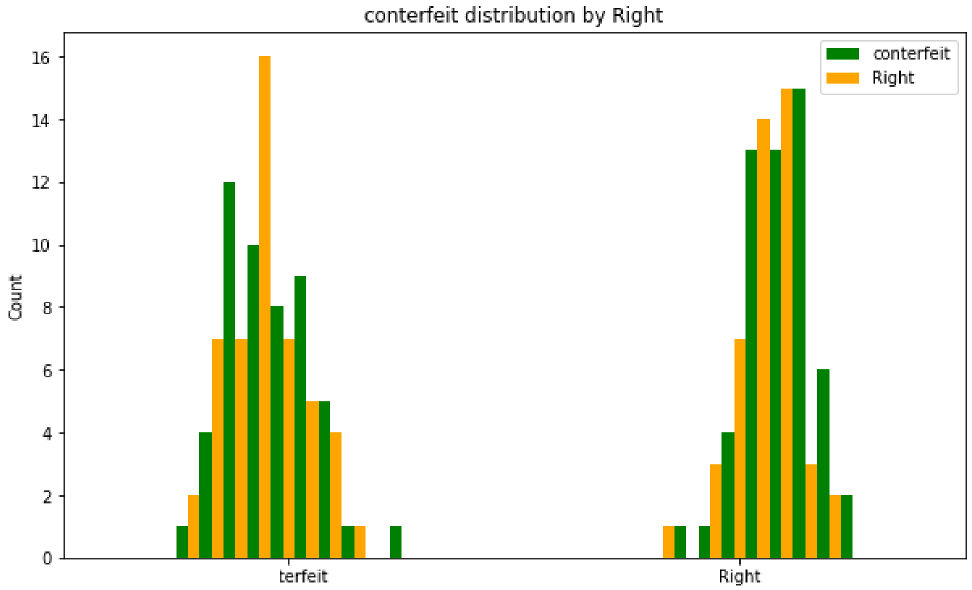


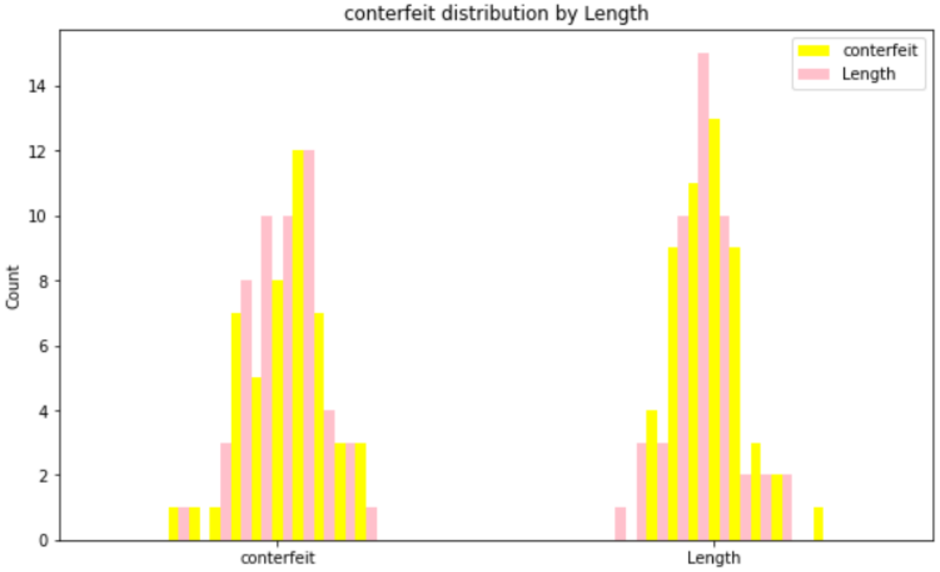




Summary


conterfeit distribution by Left
Count
conterfeit
Left


conterfeit distribution by Right
Count
conterfeit
Right
terfeit


conterfeit distribution by Length
Count
conterfeit
Length

Limitations

- No null values
- Limited data

Thank You

Team AWS