```
# -*- coding: utf-8 -*-
"""Copy of Copy of Project_Main.ipynb

Automatically
generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1VBAbhTMwx5D-rT8fwX1z9fjKRMho2_KH
"""

f
rom google.colab import drive
drive.mount('/content/drive')

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

data_path =
'/content/drive/MyDrive/data science '
train_path = os.path.join(data_path,
'train.csv')
test_path = os.path.join(data_path, 'test.csv')

my_data =
pd.read_csv(train_path)
my_data

mis_val = my_data.isnull().sum()
mis_val = mis_val[mis_val
> 0]
mis_val



after_drop = my_data.drop(columns=['SellerID',
'ExpressID'])
after_drop

col_datatype =
after_drop.dtypes
print(col_datatype)

after_drop['ExportationCountry'].unique()

after_drop.nu
nique()



after_drop.count()

# checking the unique in the column
'ProcessType'
after_drop['ProcessType'].unique()

# to know how many times each value occurs in
the 'PaymentType' column
after_drop['ProcessType'].value_counts()

# to know how many times
each value occurs in the 'PaymentType' column

after_drop['PaymentType'].value_counts()

after_drop['ProcessType'].value_counts()

##
checking the unique in the column 'ProcessType'
after_drop['DisplayIndicator'].unique()
```

```
#to
know how many times each value occurs in the 'DisplayIndicator' column

after_drop['DisplayIndicator'].value_counts()

### checking the unique in the column
'TransactionNature'
after_drop['TransactionNature'].value_counts()

#
after_drop['Type'].value_
counts()

after_drop['PaymentType'].value_counts()

after_drop['BorderTransportMeans'].value_co
unts()

after_drop['IssueDateTime'].value_counts()

after_drop['Fake'].value_counts()

after_dr
op['TaxRate'].unique()

after_drop[after_drop.TaxRate ==
'0.000e+00']

sns.countplot(x='TaxRate', hue='Fake', data=after_drop)
#important

final_data =
after_drop.copy()
final_data.corr()

final_data['IssueDateTime'] =
pd.to_datetime(final_data['IssueDateTime']).dt.month

final_data['ClassificationID'] =
final_data['ClassificationID'].astype(str)
final_data.loc[final_data['ClassificationID'].str.le
n() == 10, 'ClassificationID'] =
final_data['ClassificationID'].str[:2]
final_data.loc[final_data['ClassificationID'].str.len()
== 9, 'ClassificationID'] = final_data['ClassificationID'].str[0]

encode_label =
['DutyRegime', 'ClassificationID', 'ExportationCountry', 'OriginCountry']

col_unique_dict =
{col: list(final_data[col].unique()) for col in encode_label}

encode_target =
['IssueDateTime',

'DisplayIndicator','BorderTransportMeans'
,'PaymentType','DeclarationOfficeID']
for col in
encode_target:
  final_data[col] =
final_data.groupby(col)["Fake"].transform("mean")
for col in encode_label:

 final_data[col] = pd.factorize(final_data[col], sort=True)[0]

col_num_dict = {col:
list(final_data[col].unique()) for col in encode_label}

dropped_cols = ['TransactionNature',
'DeclarerID', 'ImporterID', 'Type', 'ProcessType']
final_data =
final_data.drop(columns=dropped_cols)

feature = final_data.columns[1:-1]
```

```
labl =
'Fake'
feature

from sklearn.model_selection import train_test_split
X_train, X_test, y_train,
y_test = train_test_split(final_data[feature], final_data[labl], test_size=0.2,
random_state=42)

from sklearn.model_selection import GridSearchCV
from sklearn.neighbors
import KNeighborsClassifier
prediction = KNeighborsClassifier()
param_search = {'n_neighbors' :
list(range(300, 305)), 'weights' : ['distance']}
grid_search = GridSearchCV(prediction ,
param_search, cv=5,
scoring='accuracy',return_train_score=True)
grid_search.fit(final_data[feature],
final_data[labl])

res = grid_search.cv_results_
params_score_list = [(params, outcome) for
(params, outcome) in zip(res['params'], res['mean_test_score'])]
outcomes = [outcome for
(params, outcome) in params_score_list] # get the scores for plotting with neighbours
neighbors
= [params['n_neighbors'] for (params, outcome) in zip(res['params'], res['mean_test_score'])] #
get the neighbors for plotting with scores
for (params, outcome) in params_score_list:

print('K:', params['n_neighbors'], ': Accuracy:', outcome)

av_score =
np.mean(outcomes)

grid_search.best_estimator_,
grid_search.best_score_

params_score_list

outcomes

neighbors

outcomes

# neighbors versus
accuracy
plt.plot(neighbors, outcomes)
plt.title('K versus score
plot')
plt.xlabel('K')
plt.ylabel('accuracies')
plt.show()

final_data

data_test =
pd.read_csv(test_path)
test_after = data_test.drop(columns=['SellerID',
'ExpressID'])

test_final = test_after.copy()
test_final['IssueDateTime'] =
pd.to_datetime(test_final['IssueDateTime']).dt.month
test_final['ClassificationID'] =
(test_final['ClassificationID'] // (1e8)).astype(int)

for col in encode_target:


test_final[col] = final_data.groupby(col)["Fake"].transform("mean")
for col
in encode_label:
 test_final[col] = test_final[col].apply(lambda x:
col_num_dict[col][col_unique_dict[col].index(x)] if x in col_unique_dict[col] else
-1)

test_final = test_final.drop(columns=dropped_cols)

test_final.columns



feature =
test_final.columns[1:]
predic =
grid_search.best_estimator_.predict(test_final[feature])

result = pd.DataFrame({'ID':
test_final['ID'], 'Fake': prediction})
filename = f"result_{prediction}_{int(av_score *
100)}.csv"
output_directory = "/content/drive/MyDrive/data science "
output_path
= os.path.join(output_directory, filename)
result.to_csv(output_path, index=False)
```