```
# load necessary libraries

import time
import pandas as pd
import pickle as pk
import numpy as np
import os
from datetime import datetime

# clustering
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn.cluster import AgglomerativeClustering

# visualization
import seaborn as sns
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

start_time = time.time()

# declare contants
kfold = 10
base_dir = '~\\project\\ExploratoryDataAnalysis'
excel_file = 'aiml_test_data.xlsx'
filename = os.path.join(base_dir, excel_file)

#
----------------------------------------------------------------------
---
# Helper modules for Descriptive Statistics
#
----------------------------------------------------------------------
---
def get_redundant_pairs(df):
        pairs_to_drop = set()
        cols = df.columns
        for i in range(0, df.shape[1]):
            for j in range(0, i+1):
                pairs_to_drop.add((cols[i], cols[j]))
        return pairs_to_drop

def get_top_abs_correlations(df, n=5):
        au_corr = df.corr().unstack()
        labels_to_drop = get_redundant_pairs(df)
        au_corr =
au_corr.drop(labels=labels_to_drop).sort_values(ascending=False)
        return au_corr[0:n]
```

```
def corrank(X):
        import itertools
        df = pd.DataFrame([[(i,j),
                   X.corr().loc[i,j]] for i,j in
list(itertools.combinations(X.corr(), 2))],
                   columns=['pairs','corr'])
        print(df.sort_values(by='corr',ascending=False))
        print()

#
----------------------------------------------------------------------
---
# load dataset
#
----------------------------------------------------------------------
---
def load_dataset(filename):
    dataset = pd.read_excel(filename, sheet_name='Sheet1', header=0,
na_values='NaN')

    print(dataset.shape);    print(dataset.head(5));
print(dataset.columns)

    feature_names = ['port_of_loading', 'port_of_discharge', 'HSCODE',
'is_coc',
        'cargo_weight', 'expected_time_of_departure', 'teu']
    target = 'HSCODE2'

    return feature_names, target, dataset

# execute the function
feature_names, target, dataset = load_dataset(filename)

(10000, 8)
  port_of_loading port_of_discharge  HSCODE  is_coc  cargo_weight  \
0           KRMAS             JPNGO  585089    True     22.502334
1           KRPUN             JPNGO  200244   False     23.879217
2           KRPUS             JPTYO  159150    True      7.049077
3           KRMAS             JPKNZ  784932   False     17.018100
4           KRPUS             JPTRG  592176   False     18.331793

  expected_time_of_departure  teu  paid_amount
0                 2022-04-13    7   671.033860
1                 2023-03-12    7  1061.490580
2                 2022-09-28    1   136.779387
3                 2022-04-17    7   776.686991
4                 2022-03-13    7   907.256793
Index(['port_of_loading', 'port_of_discharge', 'HSCODE', 'is_coc',
       'cargo_weight', 'expected_time_of_departure', 'teu',
'paid_amount'],
      dtype='object')

#
----------------------------------------------------------------------
---
# find missing values in dataset if exists
#
----------------------------------------------------------------------
---
def find_missing_value(feature_names, target, dataset):
        # Count Number of Missing Value on Each Column
        print('\nCount Number of Missing Value on Each Column: ')

        print(dataset.isnull().sum(axis=0))

# execute the function
find_missing_value(feature_names, target, dataset)


Count Number of Missing Value on Each Column:
port_of_loading              0
port_of_discharge            0
HSCODE                       0
is_coc                       0
cargo_weight                 0
expected_time_of_departure   0
teu                          0
paid_amount                  0
dtype: int64

#
----------------------------------------------------------------------
---
# factorize text values & Sort by
#
----------------------------------------------------------------------
---
def factorzie_text_values(dataset):
    ports_of_loading, pol = pd.factorize(dataset['port_of_loading'])
    dataset['pol'] = pd.DataFrame(ports_of_loading)

    ports_of_discharge, pod =
pd.factorize(dataset['port_of_discharge'])
    dataset['pod'] = pd.DataFrame(ports_of_discharge)

    dataset['is_coc'] = dataset['is_coc'].astype(int)

    date_string =
dataset['expected_time_of_departure'].dt.strftime('%Y%m%d')
    dataset['date'] = date_string.astype(int)
    dataset['HSCODE2'] = (dataset['HSCODE']/10000).astype(int)

    dataset.sort_values(by=['expected_time_of_departure'], axis=0,
ascending=True, inplace=True)

    print(dataset.head(5))

    return pol, pod, dataset


pol, pod, dataset = factorzie_text_values(dataset)

     port_of_loading port_of_discharge  HSCODE  is_coc
cargo_weight  \
1149           KRPUN             JPTRG  629092       0      7.520364

7559           KRMAS             JPTYO  606212       1     15.950675

2299           KRPUN             JPOSA  286258       1     12.774403

2368           KRPUN             JPTRG  656268       1     12.935366

885            KRPUN             JPTYO  834404       0     10.691159


     expected_time_of_departure  teu  paid_amount  pol  pod      date
HSCODE2
1149                 2022-01-01    4   534.229651    1    3  20220101
62
7559                 2022-01-01    5   478.725798    0    1  20220101
60
2299                 2022-01-01    4   370.456017    1    4  20220101
28
2368                 2022-01-01    5   493.714229    1    3  20220101
65
885                  2022-01-01    6   692.056874    1    1  20220101
83

#
----------------------------------------------------------------------
---
# 1.
#
----------------------------------------------------------------------
---

# pol   ['KRMAS:    ', 'KRPUN:      ', 'KRPUS:    ', 'KRPTK:    ']
print(pol)
# pod   ['JPNGO:    ', 'JPTYO:    ', 'JPKNZ:    ', 'JPTRG:
ㅊㅡㄹㅜ가ㅎㅏㅇ', 'JPOSA:    ']
```

```
print(pod)
# HSCODE: HS CODE
#    HS CODE  6     ,         4ㅈㅏㄹㅣㄹㅡㄹ ㅊㅜ가ㅎㅐ ㅅㅏㅇㅛㅇㅎㅏㄱㅗ

# coc(Carrier Own Container):         coc   .              .
# soc(Shipper Own Container):


ㅅㅣㅈㅓㄱ ㅊㅏㅇㄱㅗ ㅇㅕㄱㅎㅏㄹㄲㅏㅈㅣ 가ㄴㅡㅇ.
# cargo_weight:
# expected_time_of_departure:
# teu(twenty-foot equivalent unit): 20                       .
#   20              ,              .
# paid_amount:

Index(['KRMAS', 'KRPUN', 'KRPUS', 'KRPTK'], dtype='object')
Index(['JPNGO', 'JPTYO', 'JPKNZ', 'JPTRG', 'JPOSA'], dtype='object')

#
----------------------------------------------------------------------
---
# descriptive statistics and correlation matrix
#
----------------------------------------------------------------------
---
def data_descriptiveStats(feature_names, target, dataset):
        # Count Number of Missing Value on Each Column
        print(); print('Count Number of Missing Value on Each Column:
')
        print(); print(dataset[feature_names].isnull().sum(axis=0))
        print(); print(dataset[target].isnull().sum(axis=0))

        # Get Information on the feature variables
        print(); print('Get Information on the feature variables: ')

        print(); print(dataset[feature_names].info())
        print(); print(dataset[feature_names].describe())

        # correlation
        print(); print(dataset[feature_names].corr())

        # Ranking of Correlation Coefficients among Variable Pairs
        print(); print("Ranking of Correlation Coefficients:")
        corrank(dataset[feature_names])

        # Print Highly Correlated Variables
        print(); print("Highly correlated variables (Absolute
Correlations):")
        print();
print(get_top_abs_correlations(dataset[feature_names], 8))
```

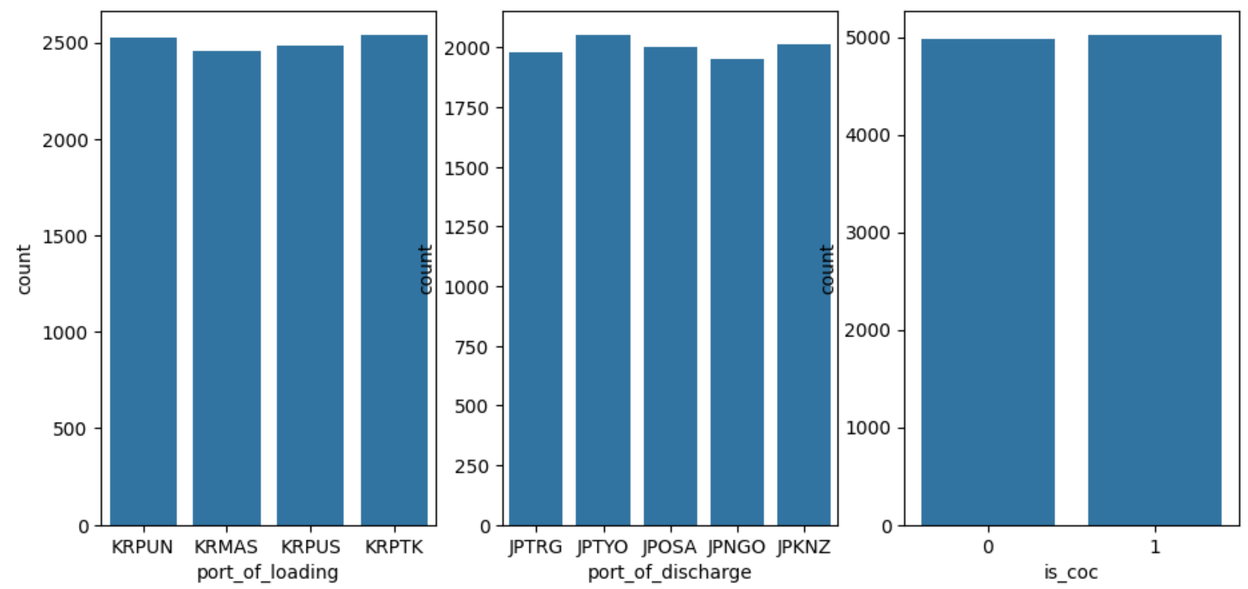
```
        # Get Information on the target
        print(); print(dataset[target].describe())
        print(); print(dataset.groupby(target).size())

feature_names = ['pol', 'pod', 'HSCODE', 'is_coc', 'cargo_weight',
'date', 'teu', 'paid_amount']
data_descriptiveStats(feature_names, target, dataset)


Count Number of Missing Value on Each Column:

pol             0
pod             0
HSCODE          0
is_coc          0
cargo_weight    0
date            0
teu             0
paid_amount     0
dtype: int64

0

Get Information on the feature variables:

<class 'pandas.core.frame.DataFrame'>
Index: 10000 entries, 1149 to 3279
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   pol           10000 non-null  int64
 1   pod           10000 non-null  int64
 2   HSCODE        10000 non-null  int64
 3   is_coc        10000 non-null  int32
 4   cargo_weight  10000 non-null  float64
 5   date          10000 non-null  int32
 6   teu           10000 non-null  int64
 7   paid_amount   10000 non-null  float64
dtypes: float64(2), int32(2), int64(4)
memory usage: 625.0 KB
None

                pol           pod         HSCODE       is_coc
cargo_weight  \
count  10000.000000  10000.000000   10000.000000  10000.00000
10000.000000
mean       1.510600      2.002900  549061.000100      0.50220
14.937857
std        1.117413      1.409074  260148.276069      0.50002
```

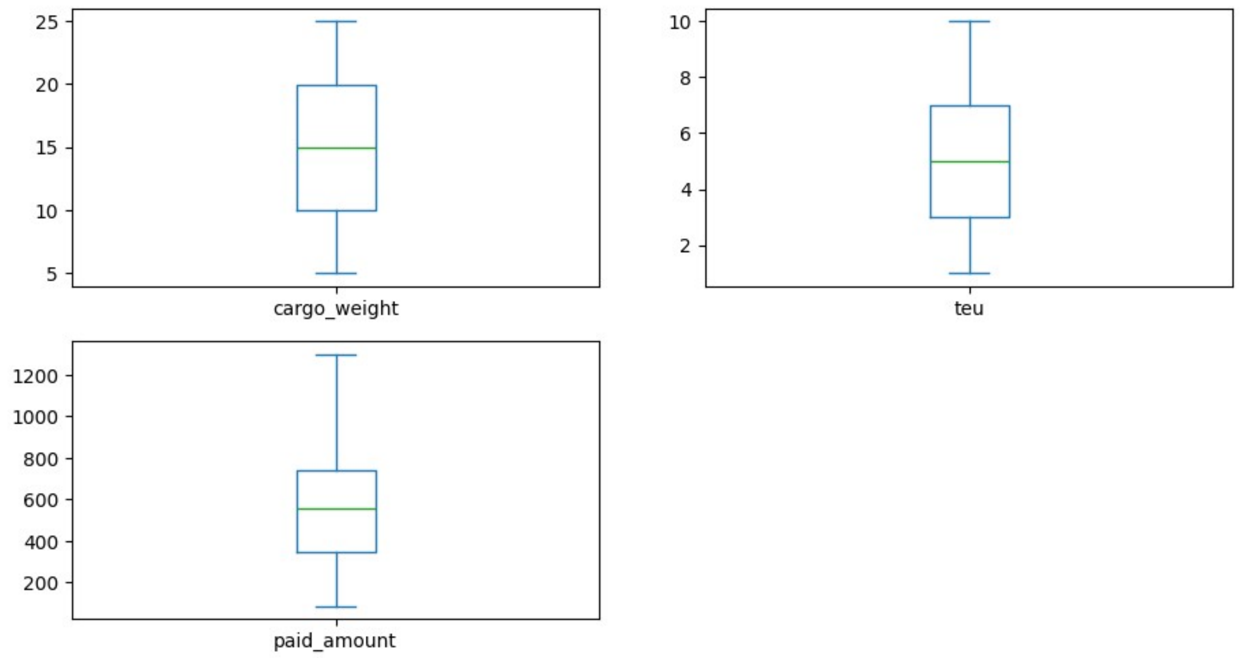
```
5.765039
min        0.000000      0.000000  100059.000000      0.00000
5.002047
25%        1.000000      1.000000  321112.000000      0.00000
10.008837
50%        2.000000      2.000000  551623.500000      1.00000
14.927572
75%        3.000000      3.000000  773352.500000      1.00000
19.921691
max        3.000000      4.000000  999908.000000      1.00000
24.997994

               date           teu   paid_amount
count  1.000000e+04  10000.000000  10000.000000
mean   2.022762e+07      4.795100    558.289845
std    6.779048e+03      2.205267    249.708996
min    2.022010e+07      1.000000     83.999559
25%    2.022073e+07      3.000000    345.581673
50%    2.023023e+07      5.000000    557.623157
75%    2.023092e+07      7.000000    739.667946
max    2.024042e+07     10.000000   1301.445152

                   pol       pod    HSCODE    is_coc  cargo_weight
date  \
pol           1.000000 -0.003227 -0.007308 -0.000579      0.005928
0.014858
pod          -0.003227  1.000000  0.011032 -0.003629      0.003381 -
0.019971
HSCODE       -0.007308  0.011032  1.000000  0.019655     -0.004936
0.004679
is_coc       -0.000579 -0.003629  0.019655  1.000000     -0.004003
0.003090
cargo_weight  0.005928  0.003381 -0.004936 -0.004003      1.000000
0.011595
date          0.014858 -0.019971  0.004679  0.003090      0.011595
1.000000
teu           0.006300 -0.003574  0.008900  0.001270      0.009588
0.192412
paid_amount   0.004216 -0.000718 -0.001401 -0.327807      0.115817
0.208216

                   teu  paid_amount
pol           0.006300     0.004216
pod          -0.003574    -0.000718
HSCODE        0.008900    -0.001401
is_coc        0.001270    -0.327807
cargo_weight  0.009588     0.115817
date          0.192412     0.208216
teu           1.000000     0.895995
paid_amount   0.895995     1.000000
```

```
Ranking of Correlation Coefficients:
                          pairs      corr
27           (teu, paid_amount)  0.895995
26          (date, paid_amount)  0.208216
25                  (date, teu)  0.192412
24  (cargo_weight, paid_amount)  0.115817
13             (HSCODE, is_coc)  0.019655
4                   (pol, date)  0.014858
22         (cargo_weight, date)  0.011595
7                 (pod, HSCODE)  0.011032
23          (cargo_weight, teu)  0.009588
16                (HSCODE, teu)  0.008900
5                    (pol, teu)  0.006300
3           (pol, cargo_weight)  0.005928
15               (HSCODE, date)  0.004679
6            (pol, paid_amount)  0.004216
9           (pod, cargo_weight)  0.003381
19               (is_coc, date)  0.003090
20                (is_coc, teu)  0.001270
2                 (pol, is_coc) -0.000579
12           (pod, paid_amount) -0.000718
17        (HSCODE, paid_amount) -0.001401
0                    (pol, pod) -0.003227
11                   (pod, teu) -0.003574
8                 (pod, is_coc) -0.003629
18       (is_coc, cargo_weight) -0.004003
14       (HSCODE, cargo_weight) -0.004936
1                 (pol, HSCODE) -0.007308
10                  (pod, date) -0.019971
21        (is_coc, paid_amount) -0.327807


Highly correlated variables (Absolute Correlations):

teu           paid_amount    0.895995
date          paid_amount    0.208216
              teu            0.192412
cargo_weight  paid_amount    0.115817
HSCODE        is_coc         0.019655
pol           date           0.014858
cargo_weight  date           0.011595
pod           HSCODE         0.011032
dtype: float64

count    10000.000000
mean        54.406500
std         26.016736
min         10.000000
25%         32.000000
```

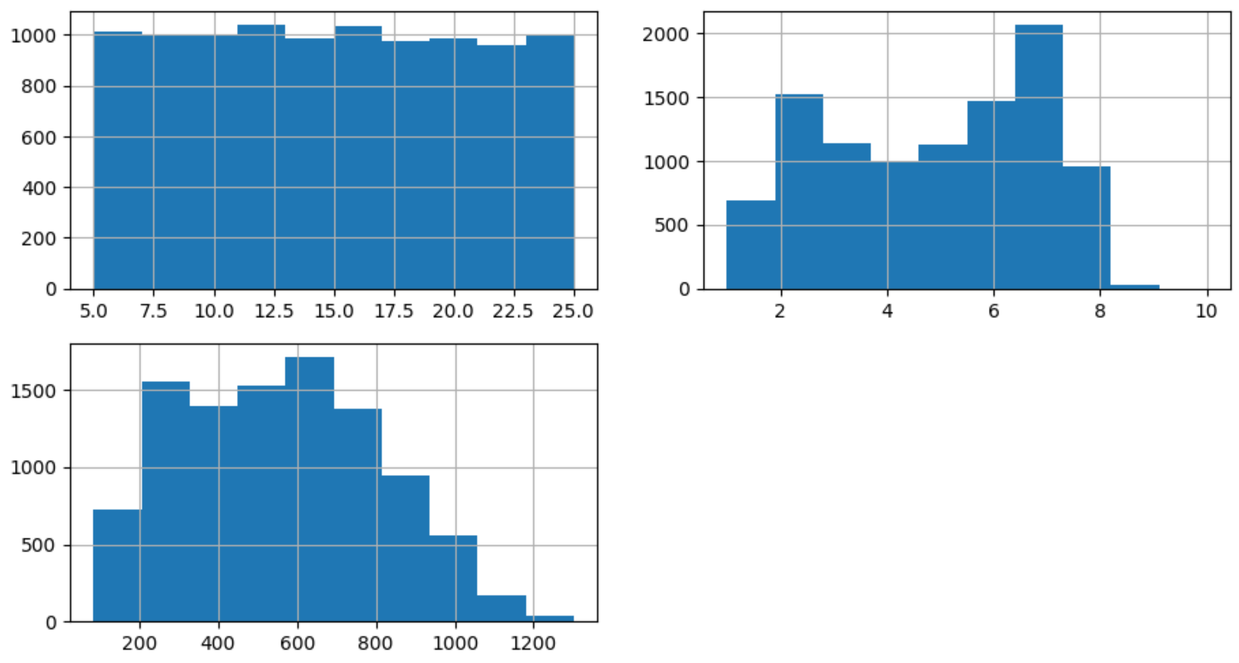
```
50%         55.000000
75%         77.000000
max         99.000000
Name: HSCODE2, dtype: float64

HSCODE2
10    124
11     99
12     98
13     94
14    100
     ...
95    122
96    109
97    125
98    121
99    113
Length: 90, dtype: int64

#
----------------------------------------------------------------------
---
# data visualisation and correlation graph
#
----------------------------------------------------------------------
---
def data_visualization(feature_names, target, dataset):
        fig, ax = plt.subplots(1,3, figsize=(11, 5))
        sns.countplot(x='port_of_loading', data=dataset, ax=ax[0])
        sns.countplot(x='port_of_discharge', data=dataset, ax=ax[1])
        sns.countplot(x='is_coc', data=dataset, ax=ax[2])
        fig.show()

        feature_names = ['cargo_weight', 'teu', 'paid_amount']
        feature_num = len(feature_names)
        # BOX plots USING box and whisker plots
        i = 1
        print(); print('BOX plot of each Numerical features')
        plt.figure(figsize=(11, 9))
        for col in feature_names:
            plt.subplot(feature_num,2,i)
            dataset[col].plot(kind='box', subplots=True, sharex=False, 
sharey=False)
            i += 1
        plt.show()

        # USING histograms
        j = 1
        print(); print('Histogram of each Numerical Feature')
        plt.figure(figsize=(11, 9))
        for col in feature_names:
            plt.subplot(feature_num,2,j)
            dataset[col].hist()
            j += 1
        plt.show()

        feature_names = ['pol', 'pod', 'HSCODE', 'is_coc',
'cargo_weight', 'date', 'teu', 'paid_amount']
        feature_num = len(feature_names)
        # correlation matrix
        print(); print('Correlation Matrix of All Numerical Features')

        fig = plt.figure(figsize=(11,9))
        ax = fig.add_subplot(111)
        cax = ax.matshow(dataset[feature_names].corr(), vmin=-1,
vmax=1, interpolation='none')
        fig.colorbar(cax)
        ticks = np.arange(0,feature_num,1)
        ax.set_xticks(ticks)
        ax.set_yticks(ticks, labels=feature_names)
        plt.show()

        # Correlation Plot using seaborn
        print(); print("Correlation plot of Numerical features")
        # Compute the correlation matrix
        corr = dataset[feature_names].corr()
        print(corr)
        # Generate a mask for the upper triangle
        mask = np.zeros_like(corr, dtype=bool)
        mask[np.triu_indices_from(mask)] = True
        # Set up the matplotlib figure
        f, ax = plt.subplots(figsize=(11, 9))
        # Generate a custom diverging colormap
        cmap = sns.diverging_palette(220, 10, as_cmap=True)
        # Draw the heatmap with the mask and correct aspect ratio
        sns.heatmap(corr, mask=mask, cmap=cmap, vmax=1.0, vmin= -1.0,
center=0, square=True,
                    linewidths=.5, cbar_kws={"shrink": .5})
        plt.show()

        # PairPlot using seaborn
        print(); print('Scatter Matrix Plot')
        sns.pairplot(dataset, hue='HSCODE2')
        plt.show()

        # Pie chart for Categorical Variables
        print(); print('PIE Chart of for Target: ')
        plt.figure(figsize=(11,9))
        i = 1
        target = ['port_of_loading', 'port_of_discharge', 'is_coc']
        for colName in target:
            labels = []; sizes = []
            df = dataset.groupby(colName).size()
            for key in df.keys():
                labels.append(key)
                sizes.append(df[key])
            # Plot PIE Chart with %
            plt.subplot(2,2,i)
            plt.axis('on')
            plt.tick_params(axis='both', left=False, top=False, 
right=False, bottom=False,
                            labelleft=True, labeltop=True, 
labelright=False, labelbottom=False)
            plt.pie(sizes, labels=labels, autopct='%1.1f%%', 
shadow=True, startangle=140)
            plt.axis('equal')
            i += 1
            # plt.savefig('Piefig.pdf', format='pdf')
        plt.show()

        # paid_amount in time series
        plt.figure(figsize=(11,7))
        plt.plot(dataset['expected_time_of_departure'], 
dataset['paid_amount'])
        plt.title("Paid Amount in time series")
        plt.xlabel("Date")
        plt.show()

data_visualization(feature_names, target, dataset)
```

```
BOX plot of each Numerical features

C:\Users\AquaCo\AppData\Local\Temp\ipykernel_13176\2147429711.py:9: 
UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be 
shown
  fig.show()
```

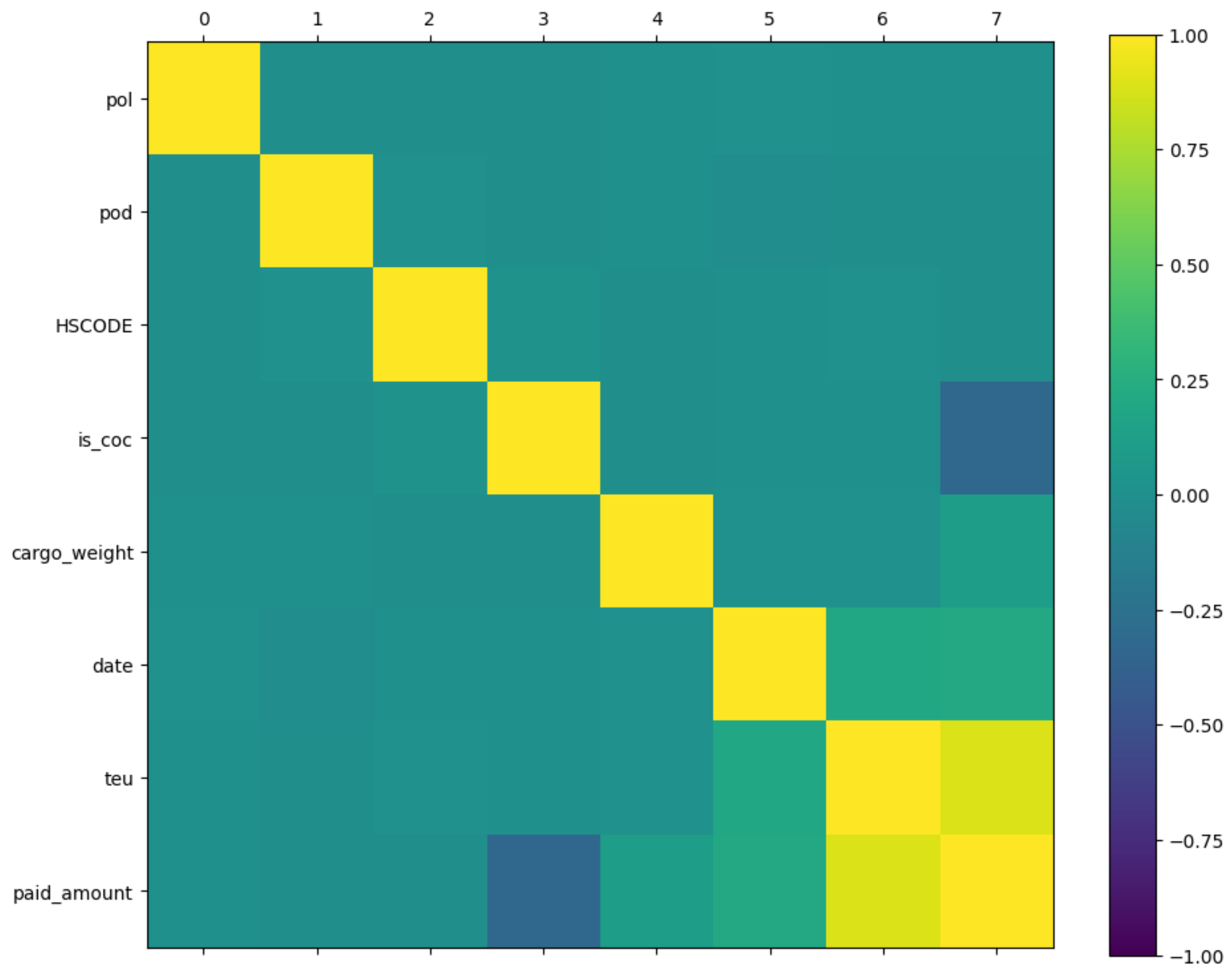




```
Histogram of each Numerical Feature
```

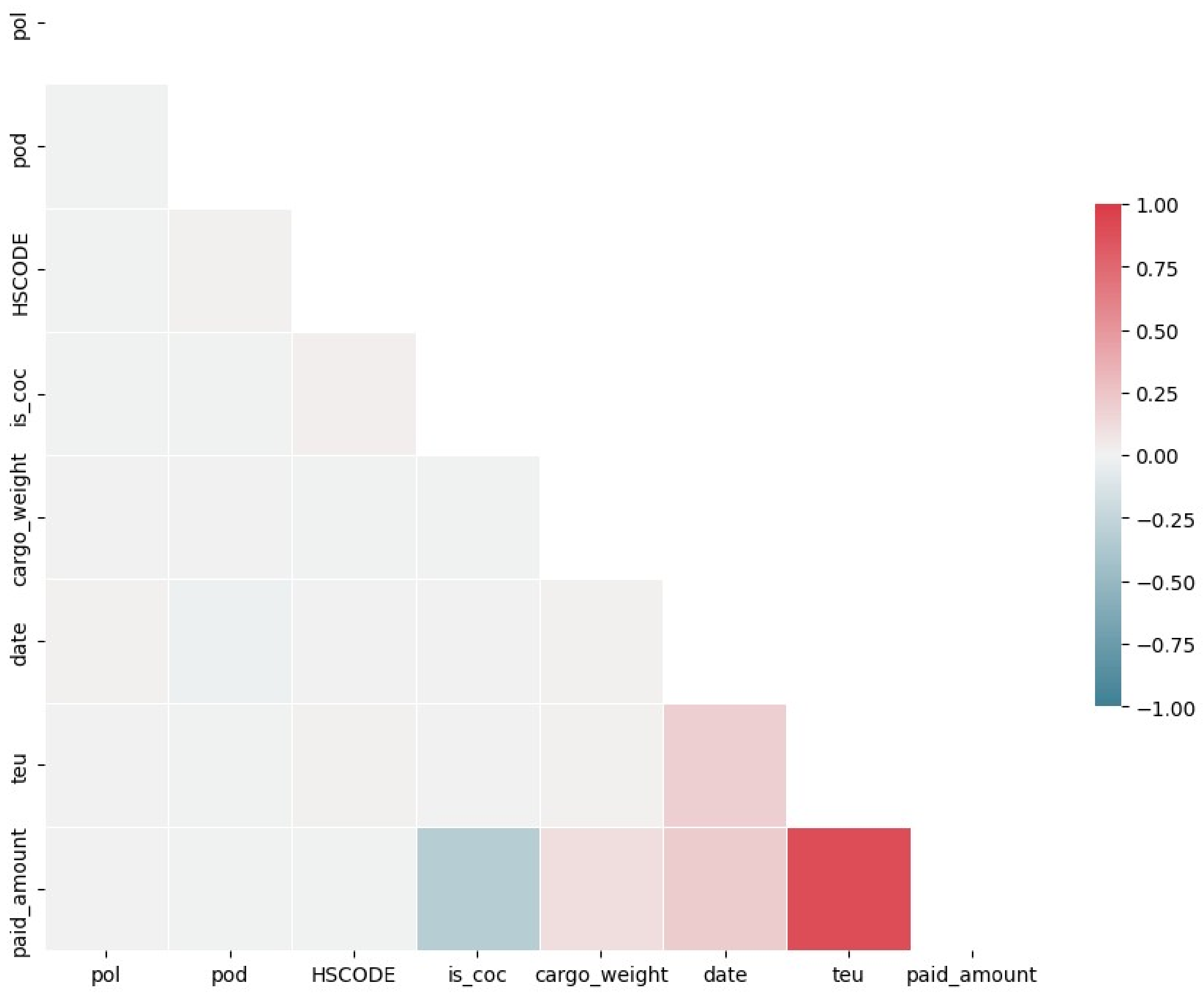


```
Correlation Matrix of All Numerical Features
```

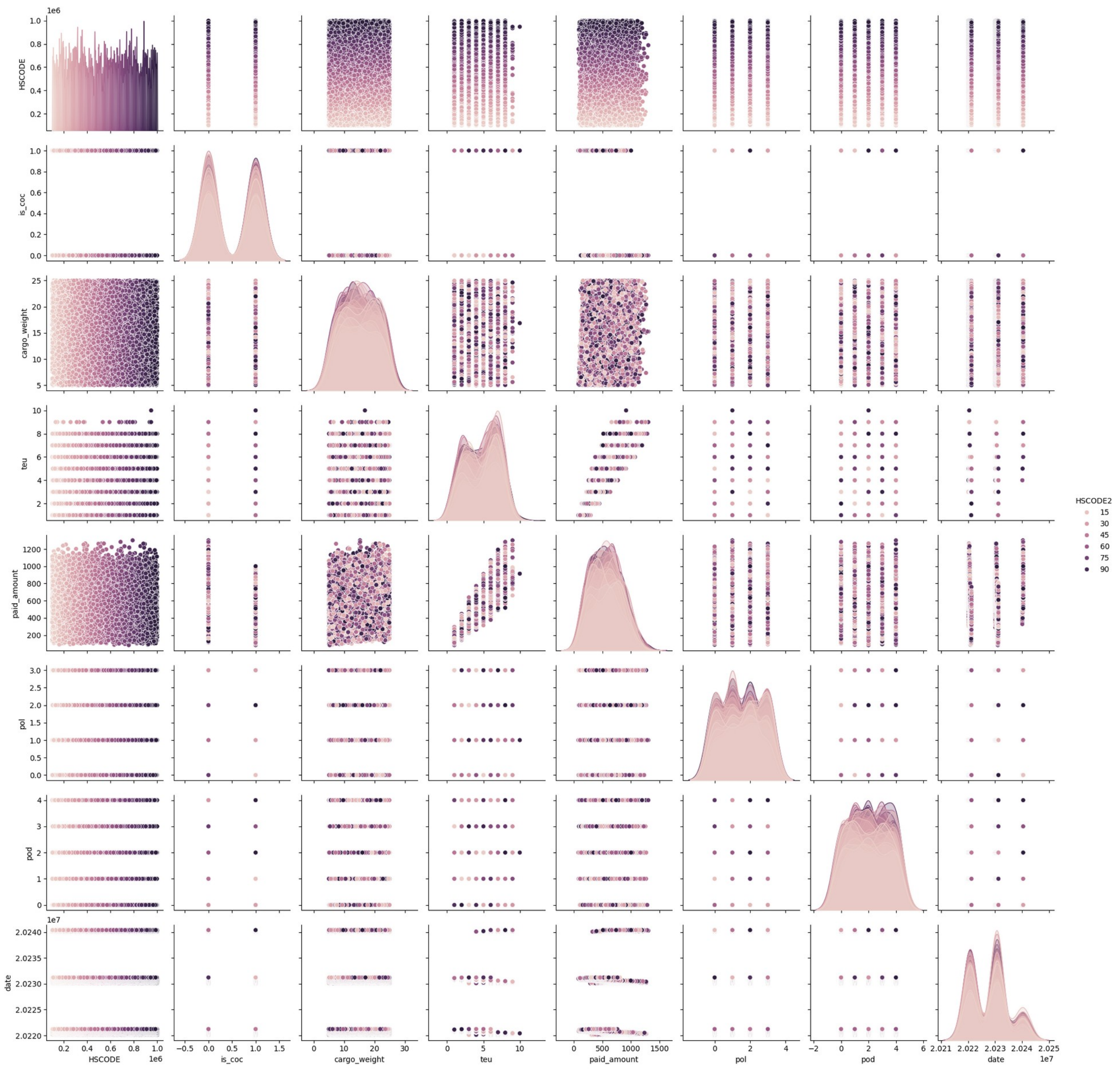


```
Correlation plot of Numerical features
                   pol       pod    HSCODE    is_coc  cargo_weight
date  \
pol           1.000000 -0.003227 -0.007308 -0.000579      0.005928
0.014858
pod          -0.003227  1.000000  0.011032 -0.003629      0.003381 -
0.019971
HSCODE       -0.007308  0.011032  1.000000  0.019655     -0.004936
0.004679
is_coc       -0.000579 -0.003629  0.019655  1.000000     -0.004003
0.003090
cargo_weight  0.005928  0.003381 -0.004936 -0.004003      1.000000
0.011595
date          0.014858 -0.019971  0.004679  0.003090      0.011595
1.000000
teu           0.006300 -0.003574  0.008900  0.001270      0.009588
0.192412
paid_amount   0.004216 -0.000718 -0.001401 -0.327807      0.115817
0.208216
```

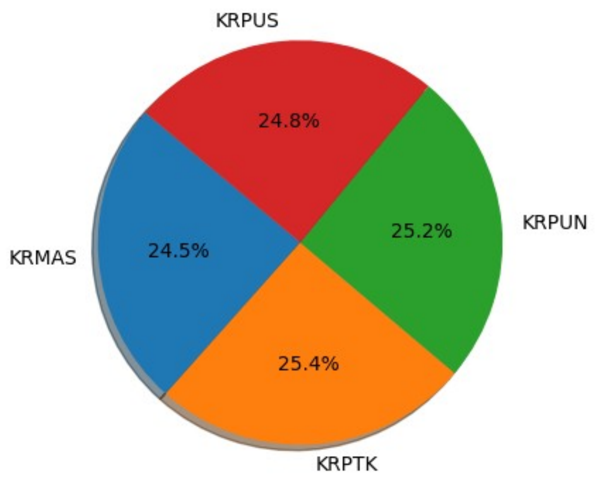
```
                  teu  paid_amount
pol          0.006300     0.004216
pod         -0.003574    -0.000718
HSCODE       0.008900    -0.001401
is_coc       0.001270    -0.327807
cargo_weight 0.009588     0.115817
date         0.192412     0.208216
teu          1.000000     0.895995
paid_amount  0.895995     1.000000
```

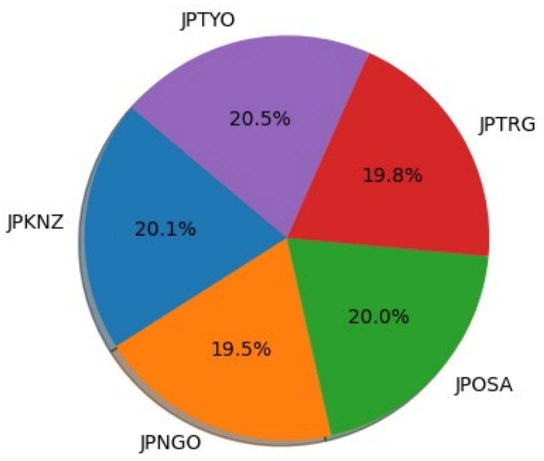


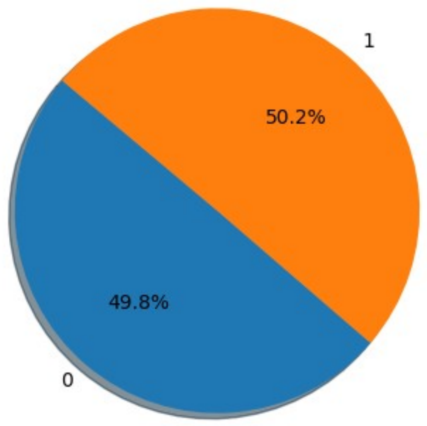
```
Scatter Matrix Plot
```

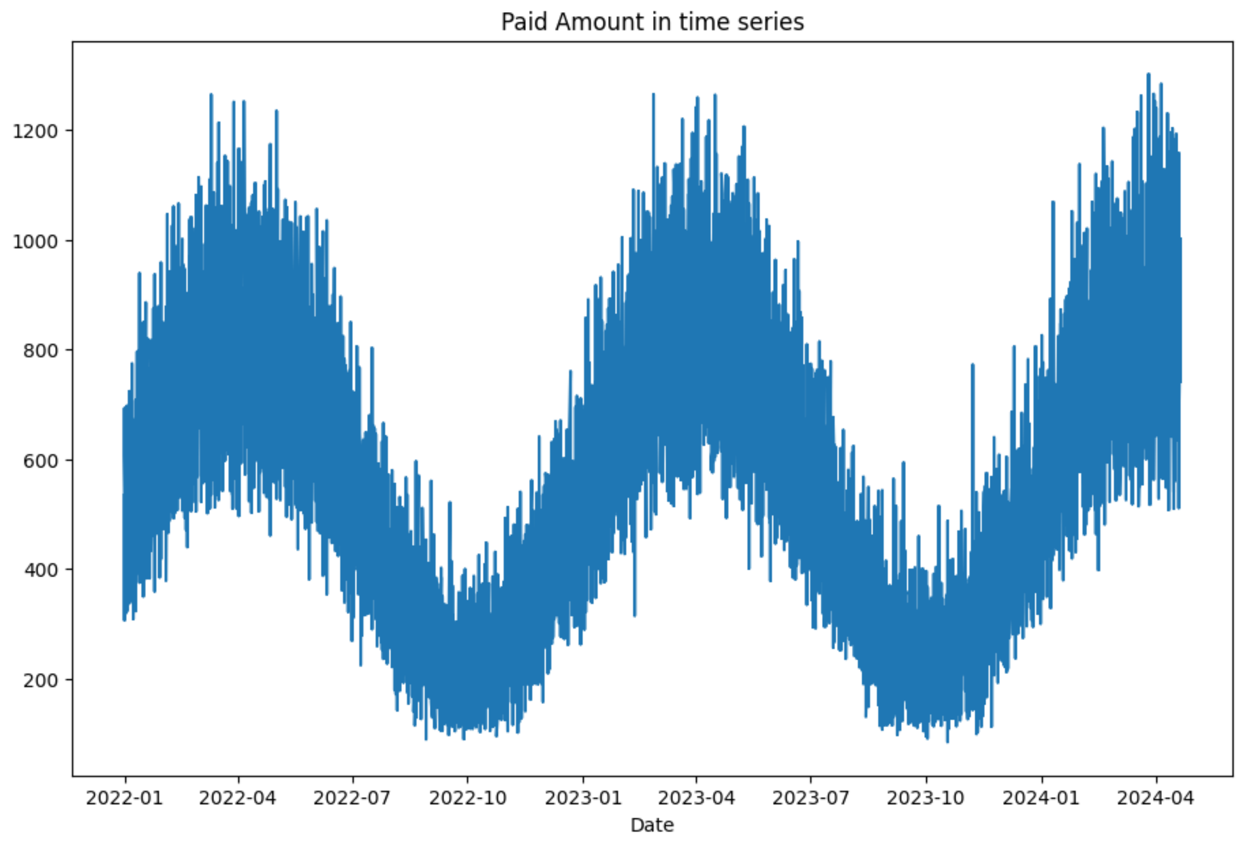


PIE Chart of for Target:


KRPUS
24.8%
KRPUN
25.2%
KRMAS
24.5%
25.4%
KRPTK


JPTYO
20.5%
JPTRG
19.8%
JPKNZ
20.1%
20.0%
19.5%
JPOSA
JPNGO

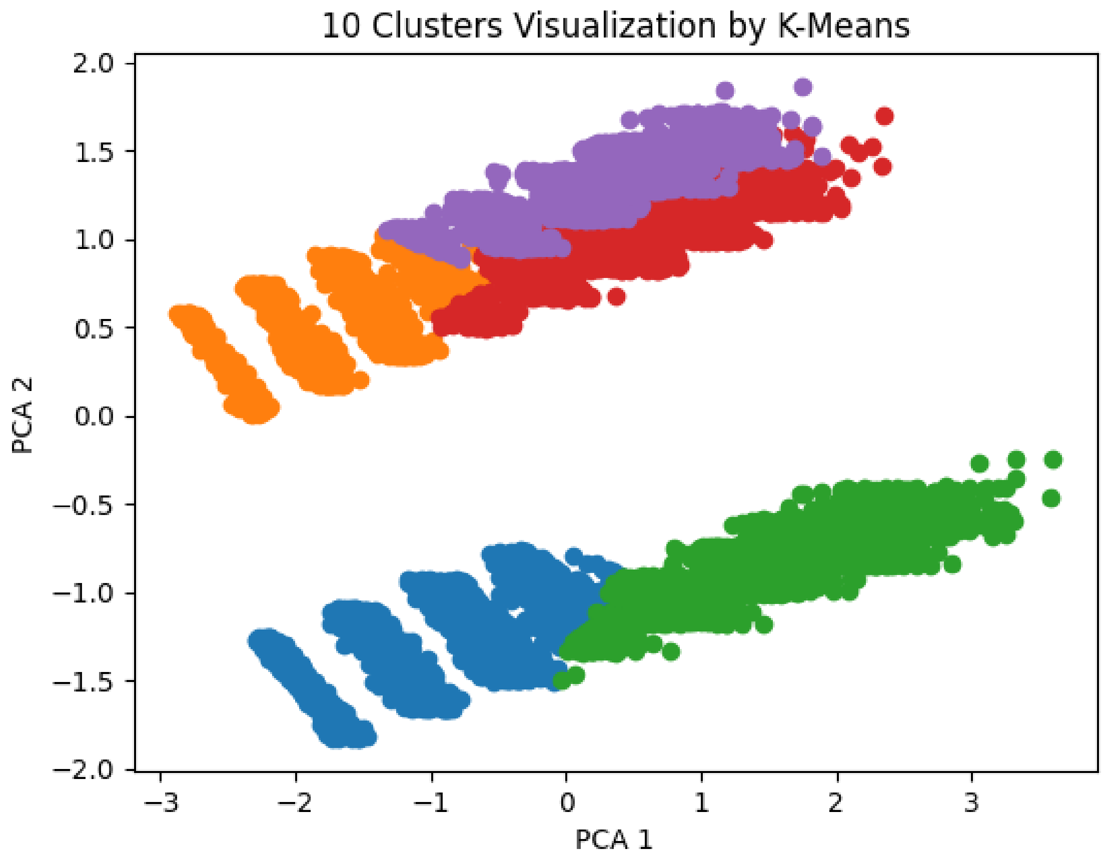

1
50.2%
49.8%
0



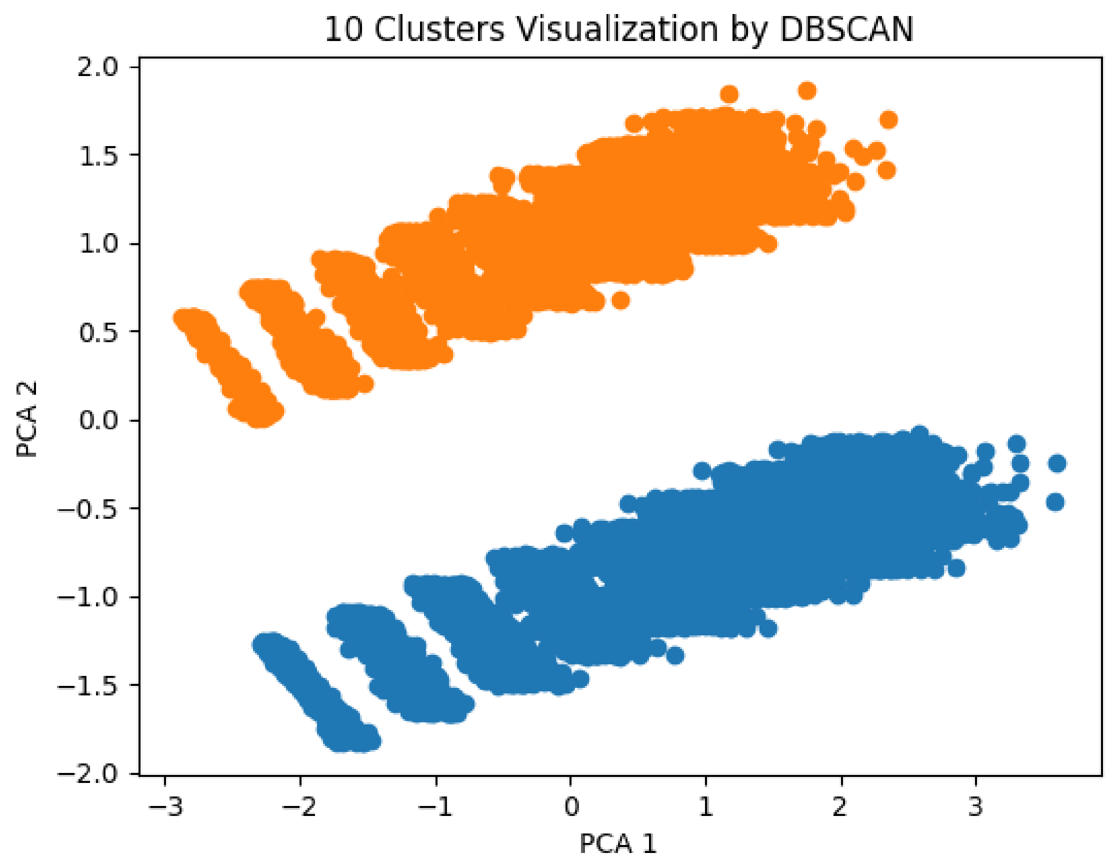
```
#
---------------------------------------------------------------------
---
# 2.1
#
---------------------------------------------------------------------
---

# 2.1.1     - K-Means, Mean Shift, DBSCAN, Agglomerative Hierarchical
Clustering

feature_names = ['is_coc', 'cargo_weight', 'teu', 'paid_amount']
X = dataset[feature_names]
X = StandardScaler().fit_transform(X)
# 4   2      PCA 2
pca = PCA(n_components=2)
pca_transformed = pca.fit_transform(X)

dataset['pca_x'] = pca_transformed[:,0]
dataset['pca_y'] = pca_transformed[:,1]

n_clusters = 6

# 2.1.1.1 KMeans
```

```
def cluster_KMeans():
    km = KMeans(n_clusters=n_clusters, init='k-means++')
    km.fit_transform(X)
    dataset['kmcluster']= km.labels_
    print('K-Means')
    print(km.labels_)

    for i in range(0,n_clusters-1):
        marker_ind = dataset[dataset['kmcluster']==i].index

        plt.scatter(x=dataset.loc[marker_ind,'pca_x'],
y=dataset.loc[marker_ind,'pca_y'])

    plt.xlabel('PCA 1')
    plt.ylabel('PCA 2')
    plt.title('10 Clusters Visualization by K-Means')
    plt.show()

cluster_KMeans()

# 2.1.1.2 DBSCAN
def cluster_DBSCAN():
    dbscan = DBSCAN(eps=0.5, min_samples=n_clusters)
    dbscan.fit(X)

    dataset['dbscancluster']= dbscan.labels_
    print('DBSCAN')
    print(dbscan.labels_)

    for i in range(0,n_clusters-1):
        marker_ind = dataset[dataset['dbscancluster']==i].index

        plt.scatter(x=dataset.loc[marker_ind,'pca_x'],
y=dataset.loc[marker_ind,'pca_y'])

    plt.xlabel('PCA 1')
    plt.ylabel('PCA 2')
    plt.title('10 Clusters Visualization by DBSCAN')
    plt.show()

cluster_DBSCAN()

# 2.1.1.3 Agglogmerative Clustering : 가ㅈㅏㅇ 가ㄲㅏㅇㅜㄴ 2
ㅁㅜㄲㅇㅓㅂㅗㅁㅕㄴㅅㅓ ㄱㅓㄹㅣㄹㅡㄹ ㄴㅡㄹㄹㅕ가ㄴㅡㄴ ㅂㅏㅇㅅㅣㄱ
def cluster_Agglogmerative():
    agg = AgglomerativeClustering(n_clusters=n_clusters)
    agg.fit(X)

    dataset['aggcluster']= agg.labels_
    print('Agglogmerative')
    print(agg.labels_)

    for i in range(0,n_clusters-1):
        marker_ind = dataset[dataset['aggcluster']==i].index

        plt.scatter(x=dataset.loc[marker_ind,'pca_x'],
y=dataset.loc[marker_ind,'pca_y'])

    plt.xlabel('PCA 1')
    plt.ylabel('PCA 2')
    plt.title('10 Clusters Visualization by Agglogmerative
Hierarchical Clustering')
    plt.show()

cluster_Agglogmerative()

# 2.1.3
#  2 cluster    Container  (is_coc)  paid_amount
#  cluster가ㅅㅓㄴㅎㅕㅇㅇㅡㄹ ㅂㅗㅇㅣㄴㅡㄴ ㅁㅗㅅㅡㅂㅇㅡㄴ teu  paid_amount
```

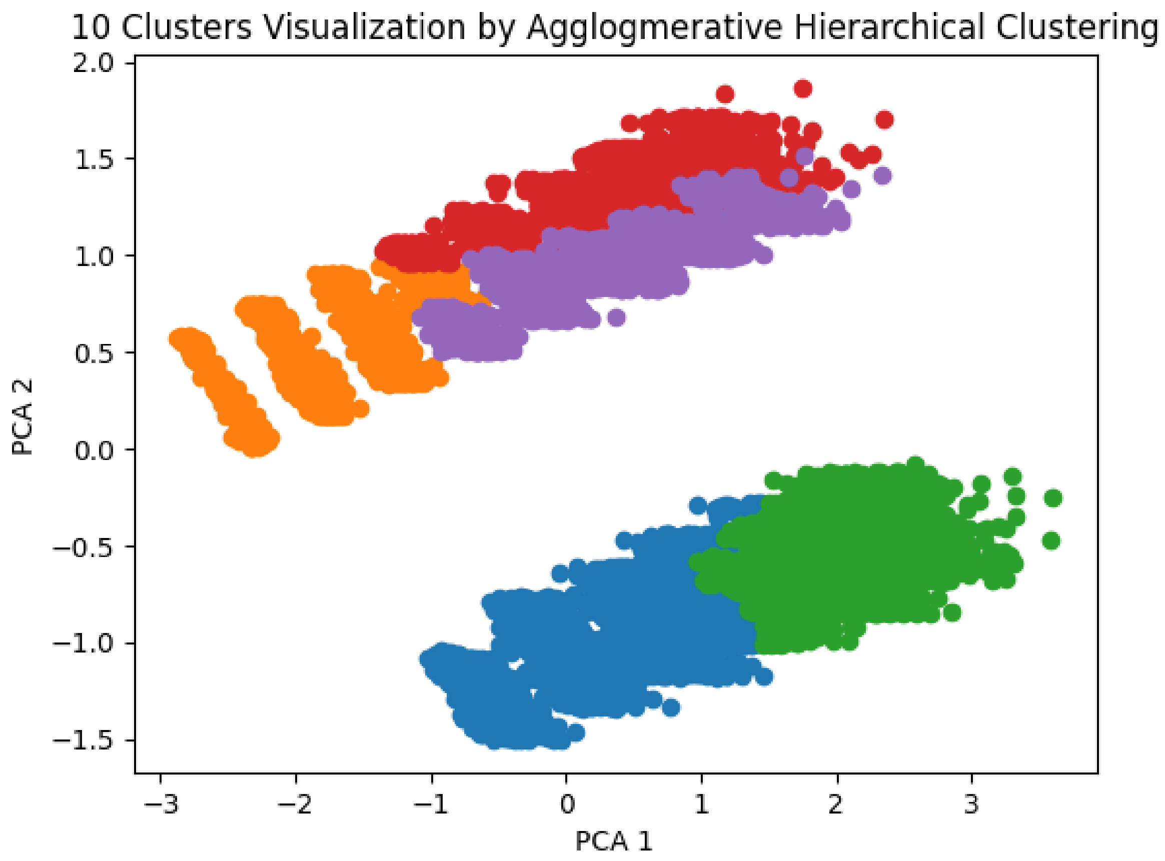
```
K-Means
[0 3 1 ... 4 3 3]
```



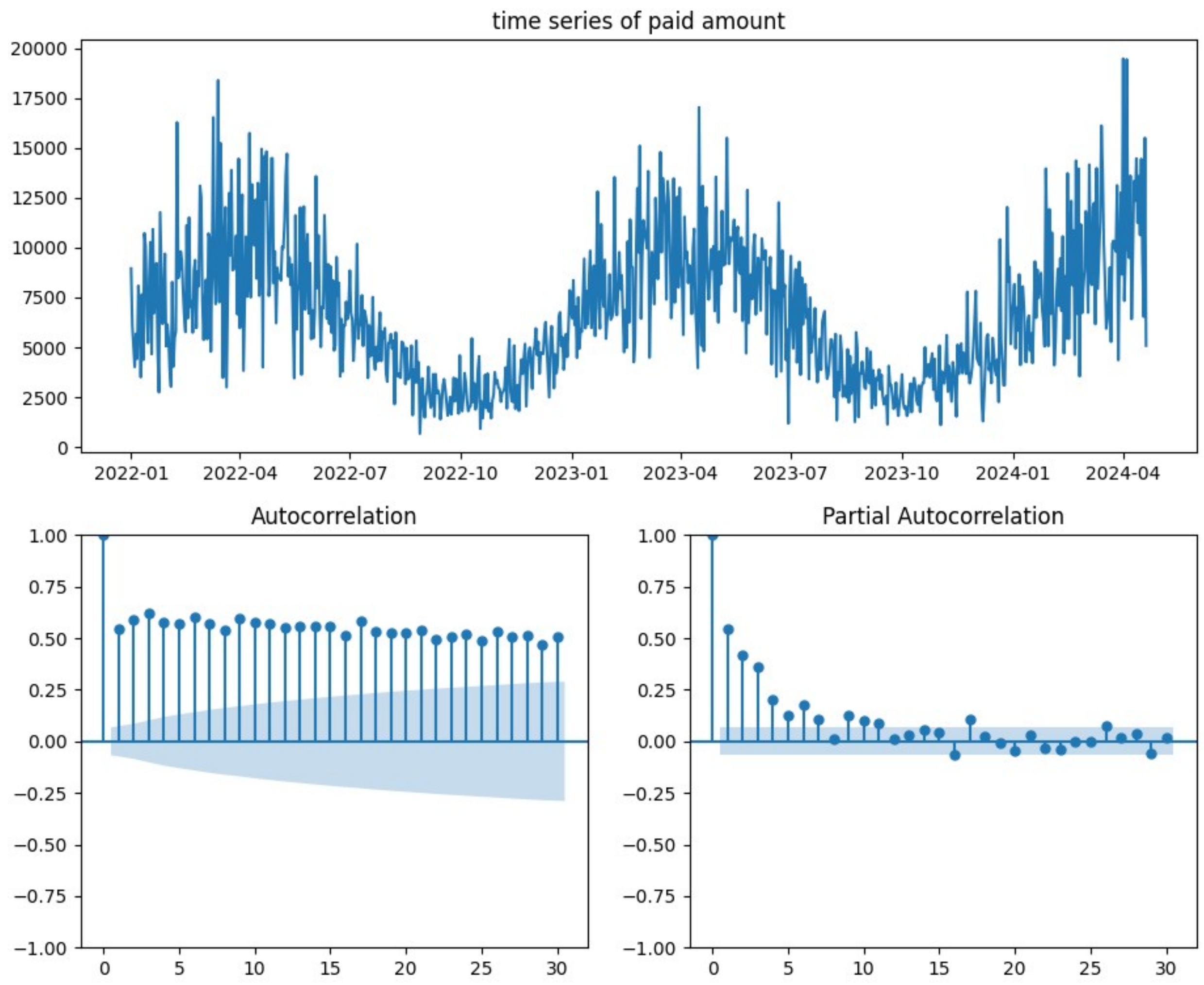
```
DBSCAN
[0 1 1 ... 1 1 1]
```

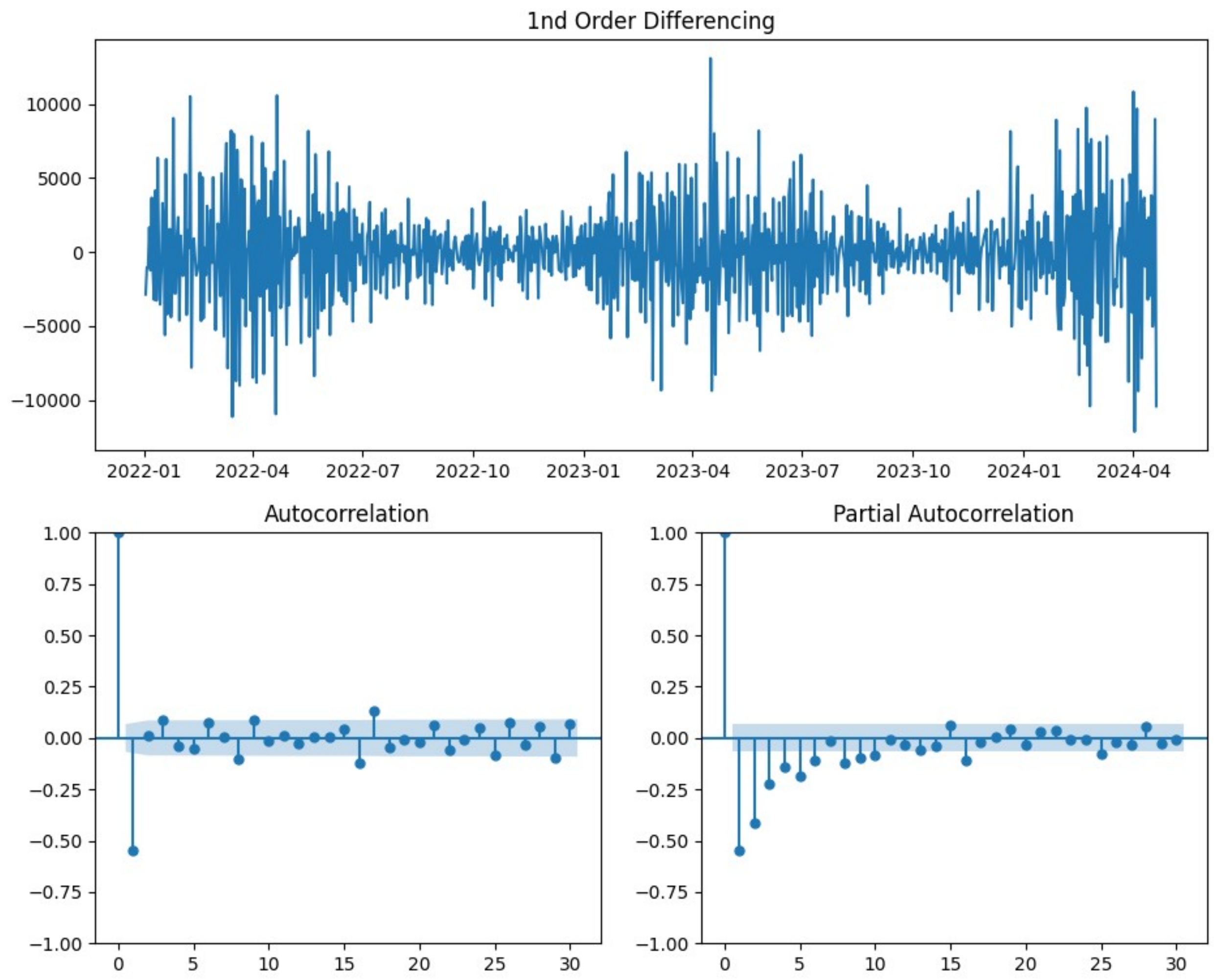


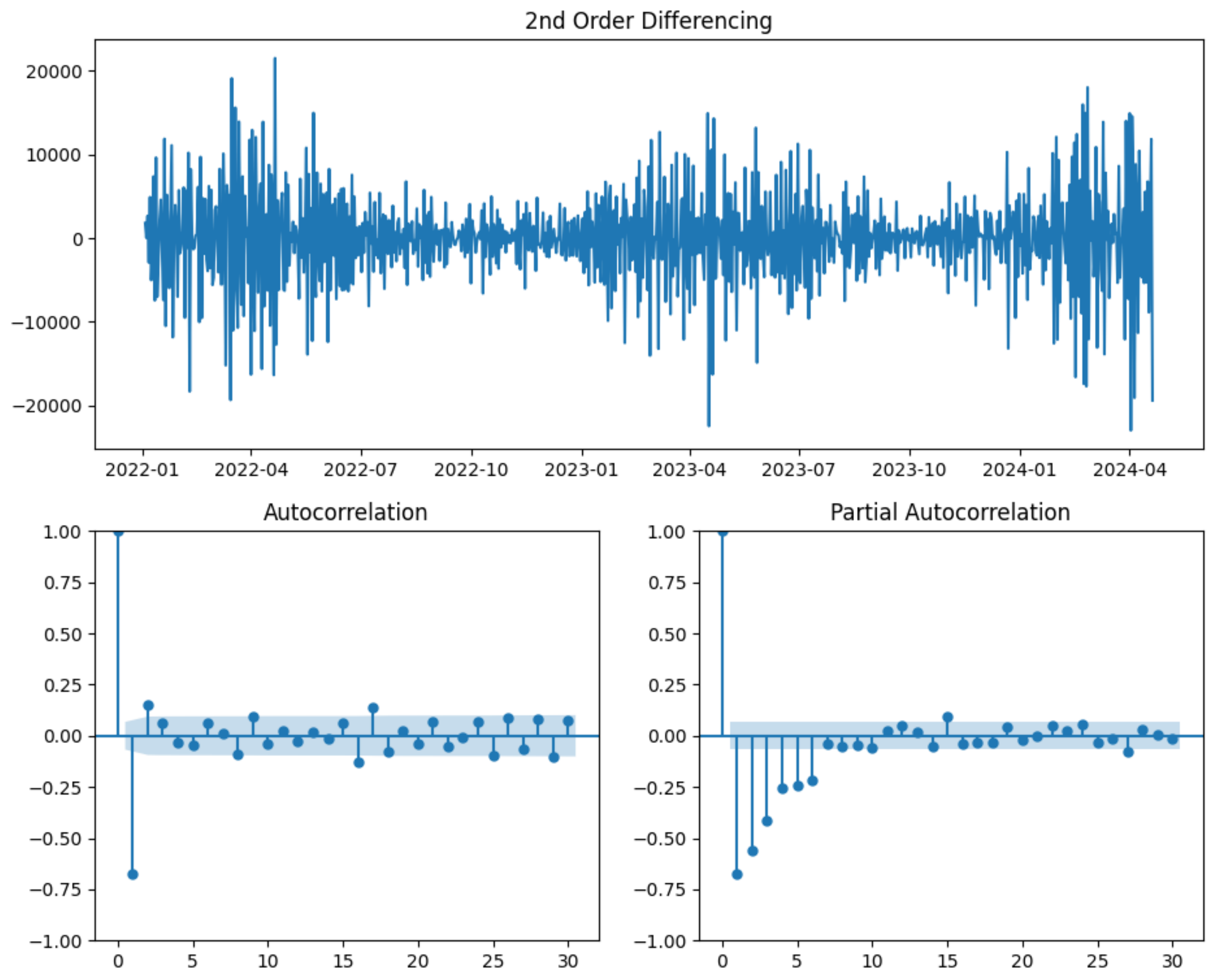
```
Agglogmerative
[0 4 1 ... 3 3 3]
```



```
#
--------------------------------------------------------------------
---
# 2.2
#
--------------------------------------------------------------------
---

ts = dataset.groupby('expected_time_of_departure')
['paid_amount'].sum()
# ACF and PACF
def acf_pacf():
    f = plt.figure(figsize=(11,9))
    ax1 = f.add_subplot(211)
    ax1.set_title('time series of paid amount')
    ax1.plot(ts)

    ax2 = f.add_subplot(223)
    plot_acf(ts, ax=ax2)
    ax3 = f.add_subplot(224)
    plot_pacf(ts, ax=ax3)
    plt.show()
    # find diff - 1st order differencing
    f = plt.figure(figsize=(11,9))
    ax11 = f.add_subplot(211)
    ax11.set_title('1nd Order Differencing')
    ax11.plot(ts.diff())

    ax12 = f.add_subplot(223)
    plot_acf(ts.diff().dropna(), ax=ax12)
    ax13 = f.add_subplot(224)
    plot_pacf(ts.diff().dropna(), ax=ax13)
    plt.show()

    # 2nd order differencing
    f = plt.figure(figsize=(11,9))
    ax21 = f.add_subplot(211)
    ax21.set_title('2nd Order Differencing')
    ax21.plot(ts.diff().diff().dropna())

    ax22 = f.add_subplot(223)
    plot_acf(ts.diff().diff().dropna(), ax=ax22)
    ax23 = f.add_subplot(224)
    plot_pacf(ts.diff().diff().dropna(), ax=ax23)
    plt.show()

acf_pacf()
```


time series of paid amount
20000
17500
15000
12500
10000
7500
5000
2500
0
2022-01
2022-04
2022-07
2022-10
2023-01
2023-04
2023-07
2023-10
2024-01
2024-04
Autocorrelation
Partial Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30


1nd Order Differencing
10000
5000
0
−5000
−10000
2022-01
2022-04
2022-07
2022-10
2023-01
2023-04
2023-07
2023-10
2024-01
2024-04
Autocorrelation
Partial Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30



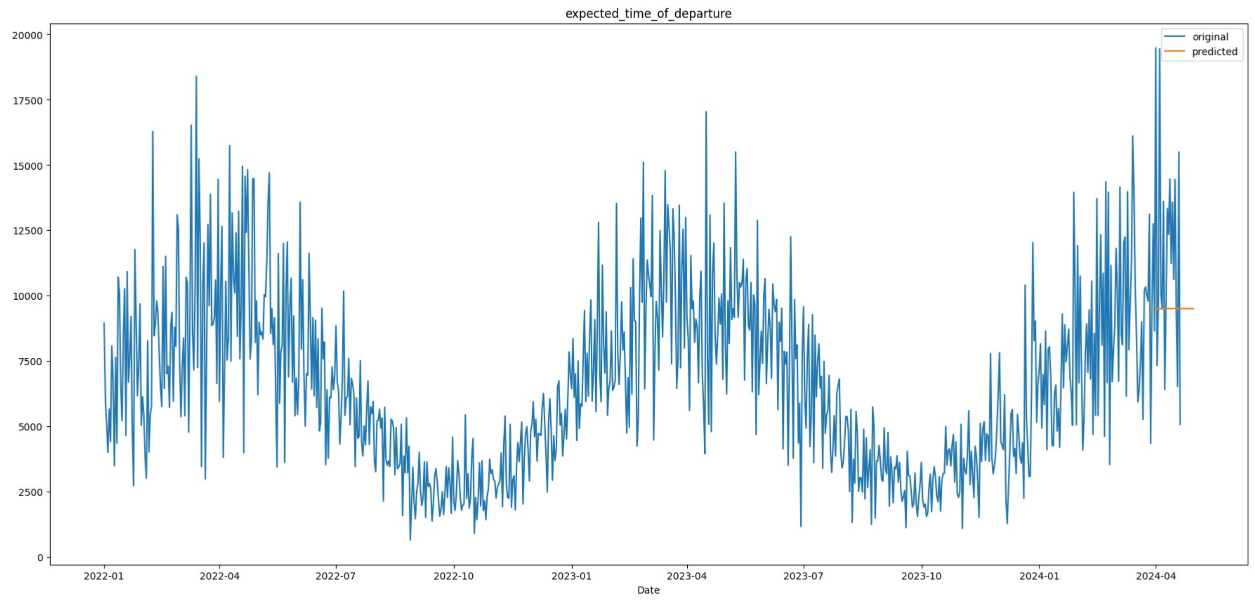
```python
from statsmodels.tsa.stattools import adfuller

result = adfuller(ts)
print('p-value: ', result[1])

result = adfuller(ts.diff().dropna())
print('p-value: ', result[1])

result = adfuller(ts.diff().diff().dropna())
print('p-value: ', result[1])
```

```
p-value:  0.7231661086972937
p-value:  1.0062369822018729e-20
p-value:  8.638952855482795e-26
```

```python
# ARIMA LIBRARY
from statsmodels.tsa.arima.model import ARIMA

# fit model
model = ARIMA(ts[ts.index < datetime(2024,4,1)], order=(1,1,2))
```

```
model_fit = model.fit()

# summary of fit model
print(model_fit.summary())

# predict
forecast = model_fit.predict(start="2024-04-01", end="2024-04-30")

# visualization
plt.figure(figsize=(22,10))
plt.plot(ts, label = "original")
plt.plot(forecast, label = "predicted")
plt.title("expected_time_of_departure")
plt.xlabel("Date")
plt.legend()
plt.show()
```

```
c:\Python311\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473:
ValueWarning: No frequency information was provided, so inferred
frequency D will be used.
  self._init_dates(dates, freq)
c:\Python311\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473:
ValueWarning: No frequency information was provided, so inferred
frequency D will be used.
  self._init_dates(dates, freq)
c:\Python311\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473:
ValueWarning: No frequency information was provided, so inferred
frequency D will be used.
  self._init_dates(dates, freq)

                               SARIMAX Results

======================================================================
========
Dep. Variable:            paid_amount   No. Observations:
821
Model:                 ARIMA(1, 1, 2)   Log Likelihood               -
7484.263
Date:                Sun, 28 Apr 2024   AIC
14976.527
Time:                        22:31:46   BIC
14995.364
Sample:                    01-01-2022   HQIC
14983.755
                         - 03-31-2024

Covariance Type:                  opg

======================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------
--------
ar.L1          0.1286      0.248      0.518      0.604      -0.358
0.615
ma.L1         -1.1369      0.241     -4.714      0.000      -1.610
-0.664
ma.L2          0.2412      0.216      1.118      0.264      -0.182
0.664
sigma2      5.273e+06   2.04e+05     25.830      0.000    4.87e+06
5.67e+06
======================================================================
=============
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
133.49
Prob(Q):                              0.99   Prob(JB):
0.00
Heteroskedasticity (H):               0.65   Skew:
0.47
Prob(H) (two-sided):                  0.00   Kurtosis:
4.74
======================================================================
=============

Warnings:
[1] Covariance matrix calculated using the outer product of gradients
(complex-step).
```



```python
# predict all path
from sklearn.metrics import mean_squared_error
```

```python
# fit model
model2 = ARIMA(ts, order=(1,1,2)) # (ARMA) = (1,0,1)
model_fit2 = model2.fit()
forecast2 = model_fit2.predict()
error = mean_squared_error(ts, forecast2)
print("error: " ,error)
# visualization
plt.figure(figsize=(22,10))
plt.plot(ts, label = "original")
plt.plot(forecast2,label = "predicted")
plt.title("Time Series Forecast")
plt.xlabel("Date")
plt.ylabel("Mean Temperature")
plt.legend()
plt.savefig('graph.png')

plt.show()

# -------------------
# save the model
# -------------------
def save_model(model):
        with open('paid_amount_model.pickle', 'wb') as f:
            pk.dump(model, f)

save_model(model)

# ------------------------------------------------
# Load the model from disk and make predictions
# ------------------------------------------------
def final_prediction(feature_names, filename):
        # load model
        f = open('paid_amount_model.pickle', 'rb')
        model = pk.load(f); f.close()

        # load dataset
        dataset = pd.read_excel(filename, sheet_name='Sheet1',
header=0, na_values='NaN')


final_prediction(feature_names, filename)

print()
print("Required Time %s seconds: " % (time.time() - start_time))
```