```
import sys
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.simplefilter('ignore')

import numpy as np


df=pd.read_csv('/content/drive/MyDrive/akadelivers/train.csv', index_col=0)
df.head()
```

order_id	local_time	country_code	store_address	payment_status	n_of_products	products_total	fina
33446280	14:11:09	AR	55379	PAID	2	11.88	Deliv
33107339	11:47:41	GT	23487	PAID	2	5.20	Deliv
32960645	11:53:53	CR	62229	PAID	1	6.03	Deliv
32089564	20:15:21	ES	29446	PAID	6	6.37	Deliv
32157739	21:32:16	AR	13917	PAID	1	5.36	Canc

```
df.isnull().sum()
```

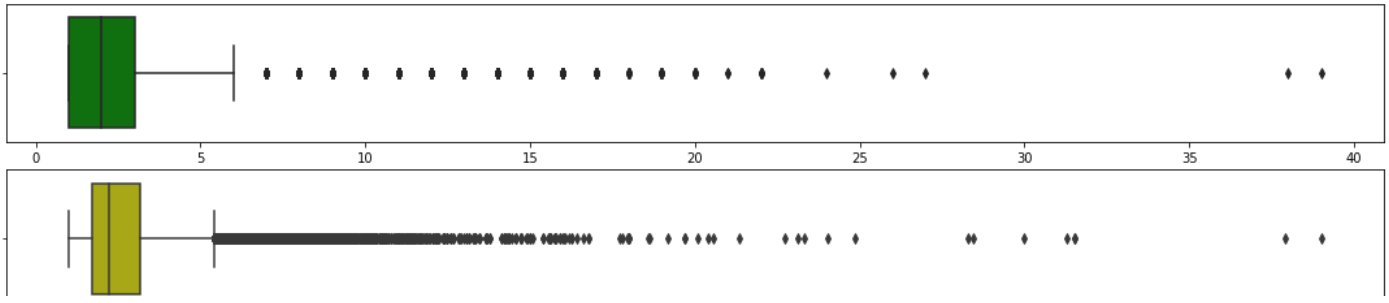
```
local_time        0
country_code      0
store_address     0
payment_status    0
n_of_products     0
products_total    0
final_status      0
dtype: int64
```

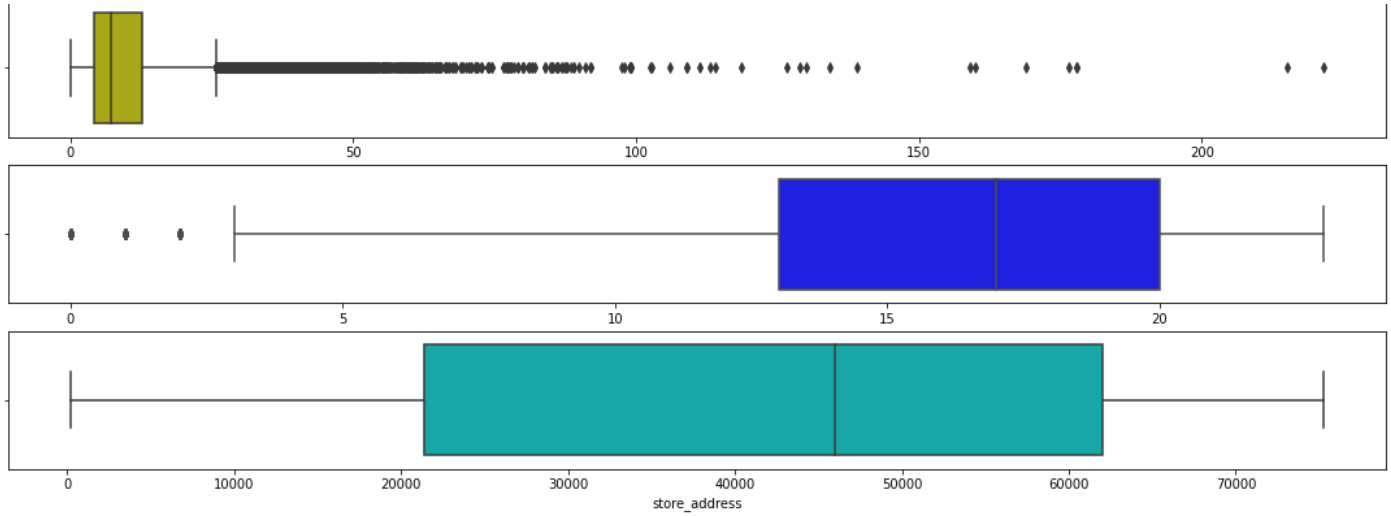
```
# convert local_time hour format
df['local_time']=df['local_time'].str[:2].astype(int)
```

Find distributions

```
fig, ax = plt.subplots(4,1,figsize=(19,9))

sns.boxplot(x=df["n_of_products"], ax=ax[0], color='g')

sns.boxplot(x=df["products_total"], ax=ax[1], color='y')

sns.boxplot(x=df["local_time"], ax=ax[2], color='b')

sns.boxplot(x=df["store_address"], ax=ax[3], color='c')
```

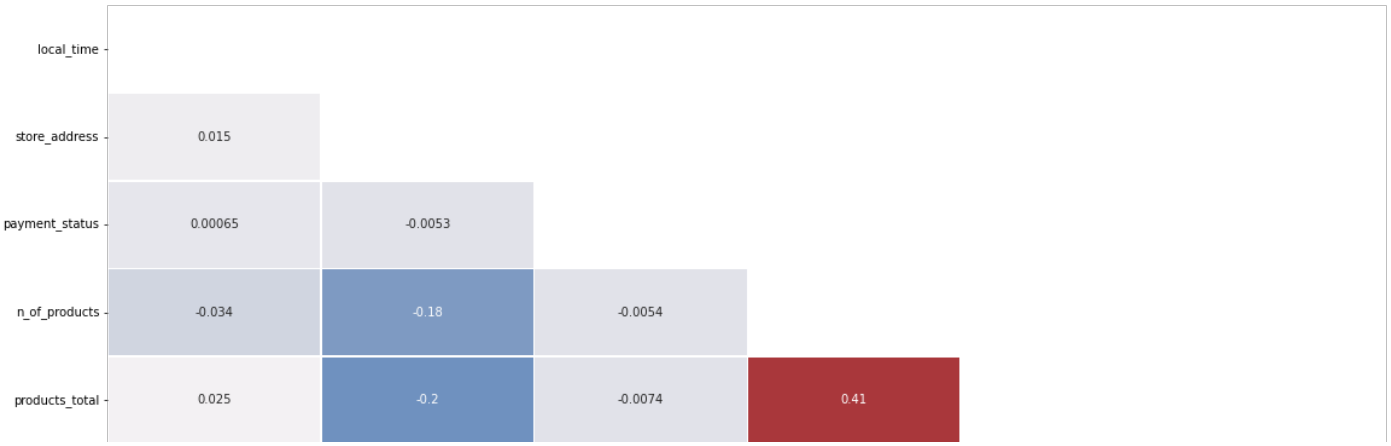
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5936ad5710>
```






```
#Transform some categorical columns to float or binary

def payment(n):
    if n == 'PAID':
        return 1
    elif n == 'NOT_PAID':
        return 0
    else:
        return 0.5

df['payment_status'] = df['payment_status'].apply(payment)

def deliver(n):
    if n == 'DeliveredStatus':
        return 1
    else:
        return 0

df['final_status'] = df['final_status'].apply(deliver)


# check correlation to discard columns if necessary
corr = df.corr()
cmap = sns.light_palette("#0c2a70", as_cmap=True)
mask = np.triu(corr)
plt.figure(figsize=(19,8))
sns.heatmap(corr, cmap='vlag', annot=True, linewidths=0.5, center=0.05, cbar=False, xticklabels=True, mask=mask)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f59369e9850>
```



Process columns

```
#cyclical features

# local_time

def sin(xxx):
  yy=np.sin((xxx)*(2.*np.pi/24))
  return yy
df['hour_sin']=df['local_time'].apply(sin)

def cos(xxx):
  yy=np.cos((xxx)*(2.*np.pi/24))
  return yy
df['hour_cos']=df['local_time'].apply(sin)

df.drop(['local_time'], axis=1, inplace=True)
```

```
from sklearn.preprocessing import MinMaxScaler, RobustScaler, OneHotEncoder

scaler = MinMaxScaler(feature_range=(0,1))
df[['store_address']] = scaler.fit_transform(df[['store_address']])

outscaler = MinMaxScaler(feature_range=(0,1))
df[['n_of_products','products_total']] = outscaler.fit_transform(df[['n_of_products','products_total']])
```

```
#Get dummies for countries

one_hot = pd.get_dummies(df['country_code'])
# Drop column B as it is now encoded
df = df.drop('country_code',axis = 1)
# Join the encoded df
df = df.join(one_hot)
```

```
df.head()
```

order_id	store_address	payment_status	n_of_products	products_total	final_status	hour_sin	hour_c
33446280	0.74	1.00	0.03	0.05	1	-0.50	-0
33107339	0.31	1.00	0.03	0.02	1	0.26	0
32960645	0.83	1.00	0.00	0.03	1	0.26	0
32089564	0.39	1.00	0.13	0.03	1	-0.87	-0

```
column_to_move = df.pop("final_status")
df.insert(len(df.columns), "final_status", column_to_move)
```

```
df['final_status'].value_counts()
```

```
1    48498
0     5832
Name: final_status, dtype: int64
```

Pre process test database

```
dftest=pd.read_csv('/content/drive/MyDrive/akadelivers/test_X.csv', index_col=0, sep=';')
dftest.head()
```

order_id	local_time	country_code	store_address	payment_status	n_of_products	products_total
32233784	17:50:09	MA	68169	PAID	1	61.63
32240990	18:38:08	ES	8220	PAID	11	15.99
33331821	22:11:59	IT	11169	PAID	4	5.89
33200505	22:13:55	AR	33371	PAID	3	7.85
32527480	12:01:04	TR	33958	PAID	2	4.75

```
# convert local_time hour format
dftest['local_time']=dftest['local_time'].str[:2].astype(int)

#Transform some categorical columns to float
def payment(n):
    if n == 'PAID':
        return 1
    elif n == 'NOT_PAID':
        return 0
    else:
        return 0.5

dftest['payment_status'] = dftest['payment_status'].apply(payment)

# cyclical features
# local_time
dftest['hour_sin']=dftest['local_time'].apply(sin)
dftest['hour_cos']=dftest['local_time'].apply(sin)

dftest.drop(['local_time'], axis=1, inplace=True)

scaler = MinMaxScaler(feature_range=(0,1))
dftest[['store_address']] = scaler.fit_transform(dftest[['store_address']])
```

order_id									
32233784	0.96	1	0.00	1.00	-0.97	-0.97	0	0	0
32240990	0.07	1	1.00	0.25	-1.00	-1.00	0	0	0
33331821	0.11	1	0.30	0.08	-0.50	-0.50	0	0	0
33200505	0.44	1	0.20	0.12	-0.50	-0.50	1	0	0
32527480	0.45	1	0.10	0.07	0.00	0.00	0	0	0

```
dftest.shape
```

```
(30, 18)
```

```
df.head()
```

	store_address	payment_status	n_of_products	products_total	hour_sin	hour_cos	AR	BR	CI
order_id									
33446280	0.74	1.00	0.03	0.05	-0.50	-0.50	1	0	0
33107339	0.31	1.00	0.03	0.02	0.26	0.26	0	0	0
32960645	0.83	1.00	0.00	0.03	0.26	0.26	0	0	0
32089564	0.39	1.00	0.13	0.03	-0.87	-0.87	0	0	0
32157739	0.18	1.00	0.00	0.02	-0.71	-0.71	1	0	0

```
df.shape
```

```
(54330, 29)
```

Apply LazyClassifier to find the bes ML model

```
import sklearn
from sklearn.utils._testing import ignore_warnings
from sklearn.model_selection import train_test_split

from sklearn import metrics
```

```
#Apply SMOTE to balance oversampling

from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)
X_sm, y_sm = sm.fit_resample(X_train, y_train)


(unique, counts) = np.unique(y_sm, return_counts=True)
frequencies = np.asarray((unique, counts)).T
frequencies
```

```
array([[   0, 7779],
       [   1, 7779]])
```

```
clf = LazyClassifier(verbose=0,ignore_warnings=True, custom_metric=None)


models,predictions = clf.fit(X_sm ,X_test, y_sm, y_test)
```

```
100%|██████████| 29/29 [02:08<00:00,  4.44s/it]
```

```
lazies=models.to_dict()


dflazy=pd.DataFrame.from_dict(lazies)


dflazy=dflazy.sort_values(by='F1 Score', ascending=False)
dflazy
```

	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
XGBClassifier	0.85	0.53	0.53	0.84	2.30
LGBMClassifier	0.86	0.51	0.51	0.84	0.50
RandomForestClassifier	0.83	0.53	0.53	0.83	2.17
BaggingClassifier	0.82	0.52	0.52	0.82	0.65
ExtraTreesClassifier	0.81	0.53	0.53	0.82	1.77
AdaBoostClassifier	0.80	0.52	0.52	0.81	0.94

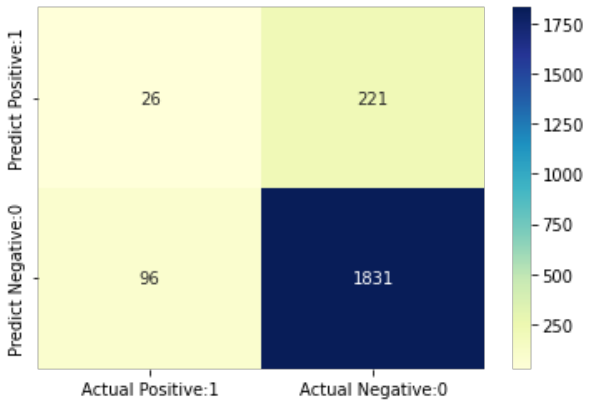
LinearSVC	0.60	0.56	0.56	0.68	3.47
NuSVC	0.59	0.51	0.51	0.67	49.75
SGDClassifier	0.55	0.53	0.53	0.64	0.22
DummyClassifier	0.50	0.52	0.52	0.59	0.04
QuadraticDiscriminantAnalysis	0.41	0.47	0.47	0.51	0.10
GaussianNB	0.11	0.49	0.49	0.04	0.05

LGBMClassifier

```
import lightgbm as ltb
```

```
fit_params={"early_stopping_rounds":30,
            "eval_metric" : 'auc',
            "eval_set" : [(X_test,y_test)],
            'eval_names': ['valid'],
            #'callbacks': [lgb.reset_parameter(learning_rate=learning_rate_010_decay_power_099)],
            'verbose': 100,
            'categorical_feature': 'auto'}
```

```
n_HP_points_to_test = 100

import lightgbm as lgb
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV

#n_estimators is set to a "large value". The actual number of trees build will depend on early stopping and 5000 define onl
clf = lgb.LGBMClassifier(max_depth=-1, random_state=314, silent=True, metric='None', n_jobs=4, n_estimators=5000)
gs = RandomizedSearchCV(
    estimator=clf, param_distributions=param_test,
    n_iter=n_HP_points_to_test,
```




```
print('F1 Score: %.3f' % f1_score(y_test, predicted_y))
print('Precision Score: %.3f' % precision_score(y_test, predicted_y))
```

```
F1 Score: 0.920
Precision Score: 0.892
```

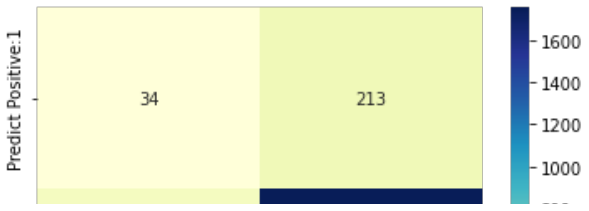
XGBClassifier

```
from xgboost import XGBClassifier
import xgboost as xgb
from sklearn.model_selection import cross_val_score, KFold


xgb_model = xgb.XGBClassifier( random_state=42)
```

Random Forest Classification

```
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(n_estimators=100)

clf.fit(X_sm,y_sm)

y_pred=clf.predict(X_test)


cm = confusion_matrix(y_test, y_pred)

cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'], index=['Predict Positive:1', 'Predict

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f593b8c2f10>
```



```
predicciones = pd.DataFrame(data=ynew)
predicciones.to_csv('predicciones.csv')
```

Extra Installs

```
!pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.7/dist-packages (0.8.1)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.7/dist-packages (from imbalan
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-packages (from imbalan
Collecting scikit-learn>=0.24
  Using cached scikit_learn-1.0.1-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (23.2 MB
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from imbalanc
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from
Installing collected packages: scikit-learn
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 0.23.1
    Uninstalling scikit-learn-0.23.1:
```