Estimação, Deteção e Análise II

06 – Deteção de outliers

Univariate

Multivariate

1. Instalar as libraries pyod, combo. No terminal:

```
pip install pyod
pip install combo
```

2. Importar as libraries necessárias

```
import pandas as pd
import seaborn as sns
from sklearn.ensemble import IsolationForest
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import matplotlib.font_manager
from sklearn.preprocessing import MinMaxScaler
from pyod.models.cblof import CBLOF
from pyod.models.feature_bagging import FeatureBagging
from pyod.models.iforest import IForest
from pyod.models.knn import KNN
from pyod.models.lof import LOF
```

3. Carregar os dados (dataset Superstore.xls disponível no Moodle)

```
df = pd.read_excel('Superstore.xls')
```

4. Visualizar estatísticas da coluna "Sales"

```
print(df['Sales'].describe())
```

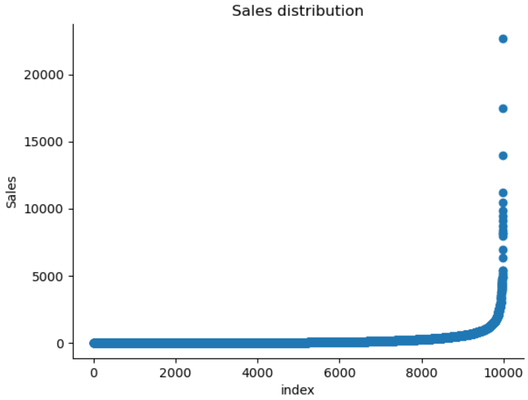
Resultado:

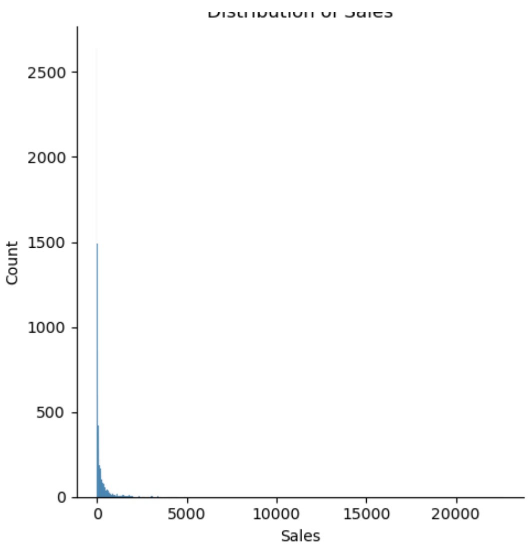
```
count     9994.000000
mean       229.858001
std        623.245101
min          0.444000
25%         17.280000
50%         54.490000
75%        209.940000
max      22638.480000
Name: Sales, dtype: float64
```

5. Visualizar distribuição de valores da coluna "Sales"

```
plt.scatter(range(df.shape[0]), np.sort(df['Sales'].values))
plt.xlabel('index')
plt.ylabel('Sales')
plt.title("Sales distribution")
sns.despine()
plt.show()

sns.displot(df['Sales'])
plt.title("Distribution of Sales")
sns.despine()
plt.show()
```

Resultado:





6. Visualizar skewness e kurtosis da coluna "Sales"

```
print("Skewness: %f" % df['Sales'].skew())
print("Kurtosis: %f" % df['Sales'].kurt())
```

Resultado:

```
Skewness: 12.972752
Kurtosis: 305.311753
```

A coluna "Sales" está longe da distribuição normal, apresentando uma cauda direita longa e estreita, estando a maior parte dos dados concentrados na parte esquerda da figura. Há menor probabilidade de aparecerem dados na parte direita da figura

7. Visualizar estatísticas da coluna "Profit"

```
df['Profit'].describe()
```

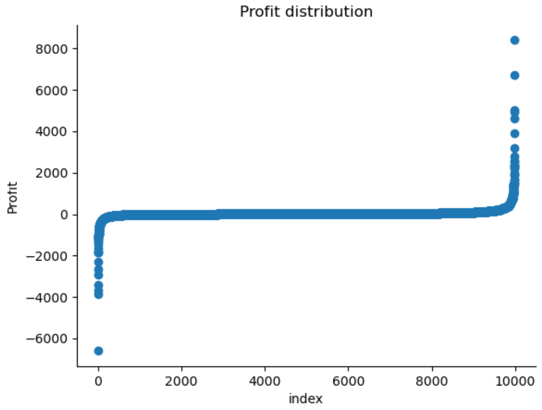
Resultado:

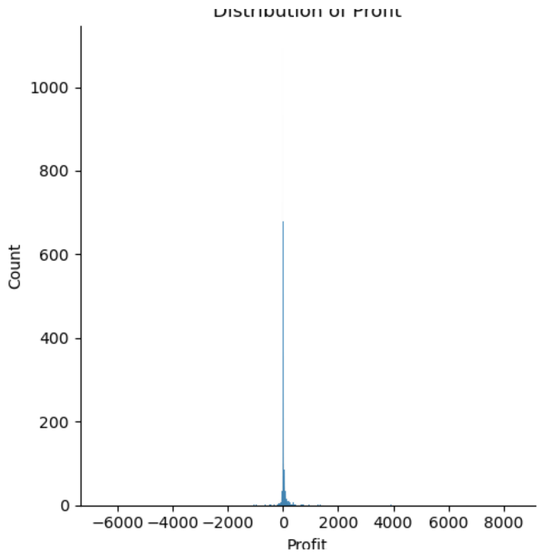
```
count     9994.000000
mean        28.656896
std        234.260108
min      -6599.978000
25%          1.728750
50%          8.666500
75%         29.364000
max       8399.976000
Name: Profit, dtype: float64
```

8. Visualizar distribuição de valores da coluna "Profit"

```
plt.scatter(range(df.shape[0]), np.sort(df['Profit'].values))
plt.xlabel('index')
plt.ylabel('Profit')
plt.title("Profit distribution")
sns.despine()
plt.show()

sns.displot(df['Profit'])
plt.title("Distribution of Profit")
sns.despine()
plt.show()
```

Resultado:





9. Visualizar skewness e kurtosis da coluna "Profit"

```
print("Skewness: %f" % df['Profit'].skew())
print("Kurtosis: %f" % df['Profit'].kurt())
```

Resultado:

```
Skewness: 7.561432
Kurtosis: 397.188515
```

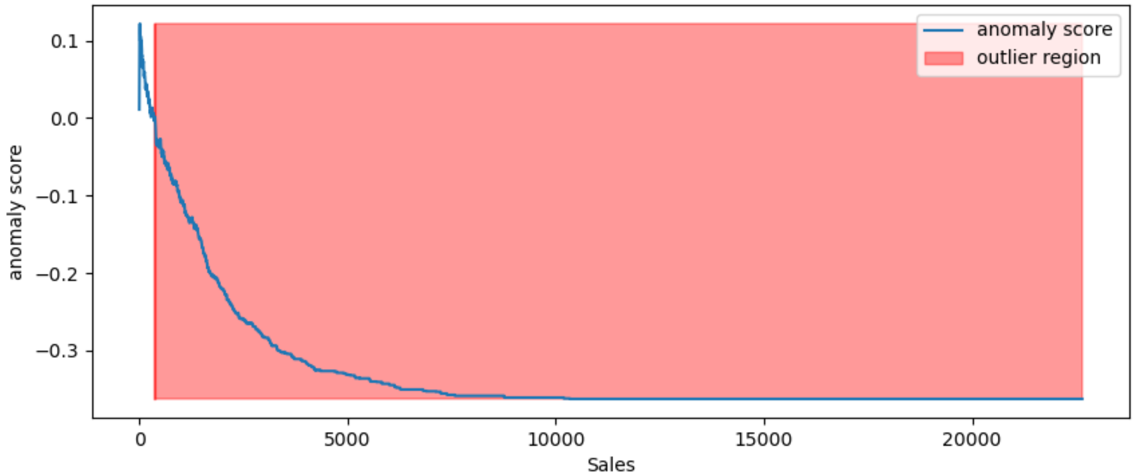
A coluna "Profit" tem caudas positiva e negativa. Há duas regiões onde podem aparecer os outliers: as pontas esquerda e direita da figura.

Univariate

10. Determinar outliers em "Sales" utilizando o algoritmo "Isolation Forest"

```
isolation_forest = IsolationForest(n_estimators=100)
isolation_forest.fit(df['Sales'].values.reshape(-1, 1))
xx = np.linspace(df['Sales'].min(), df['Sales'].max(), len(df)).reshape(-1,1)
anomaly_score = isolation_forest.decision_function(xx)
outlier = isolation_forest.predict(xx)
plt.figure(figsize=(10,4))
plt.plot(xx, anomaly_score, label='anomaly score')
plt.fill_between(xx.T[0], np.min(anomaly_score), np.max(anomaly_score),
                 where=outlier==-1, color='r',
                 alpha=.4, label='outlier region')
plt.legend()
plt.ylabel('anomaly score')
plt.xlabel('Sales')
plt.show()
```

Resultado:

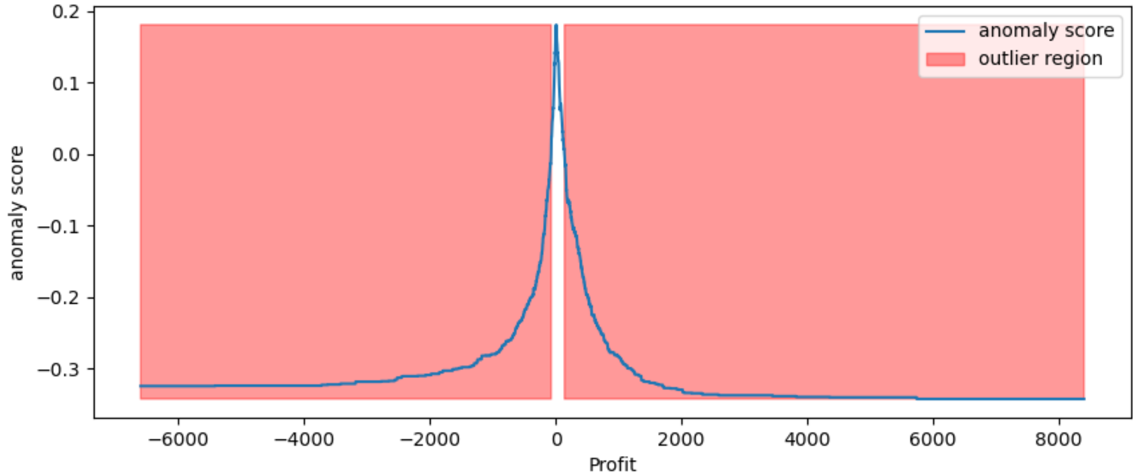


De acordo com a figura, "Sales" superiores a 1000 irão ser consideradas outliers

11. Determinar outliers em "Profit" utilizando o algoritmo "Isolation Forest"

```
isolation_forest = IsolationForest(n_estimators=100)
isolation_forest.fit(df['Profit'].values.reshape(-1, 1))
xx = np.linspace(df['Profit'].min(), df['Profit'].max(), len(df)).reshape(-1,1)
anomaly_score = isolation_forest.decision_function(xx)
outlier = isolation_forest.predict(xx)
plt.figure(figsize=(10,4))
plt.plot(xx, anomaly_score, label='anomaly score')
plt.fill_between(xx.T[0], np.min(anomaly_score), np.max(anomaly_score),
                 where=outlier==-1, color='r',
                 alpha=.4, label='outlier region')
plt.legend()
plt.ylabel('anomaly score')
plt.xlabel('Profit')
plt.show()
```

Resultado:



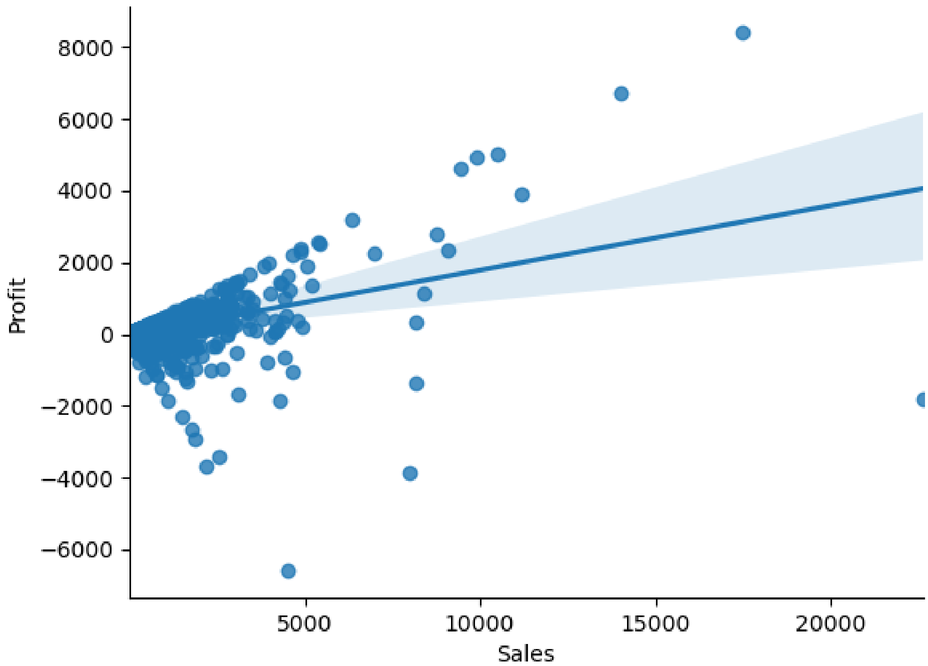
De acordo com a figura, "Profit" inferior a -100 ou superior a 100 podem ser considerados outliers

Multivariate

12. Visualizar o gráfico de correlação entre "Sales" e "Profit"

```
sns.regplot(x="Sales", y="Profit", data=df)
sns.despine()
plt.show()
```

Resultado:



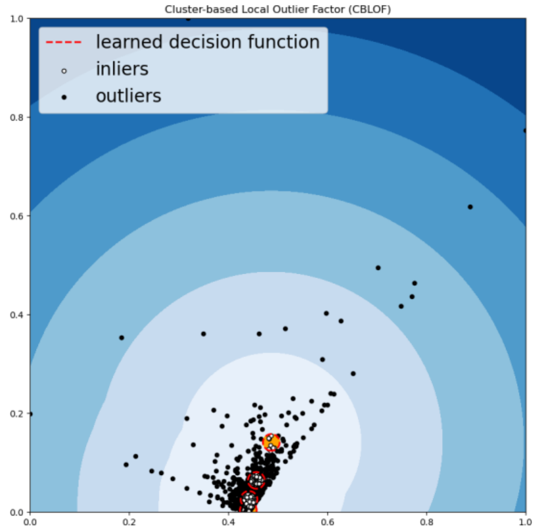
Conseguimos detetar alguns outliers - valores demasiado altos ou demasiado baixos

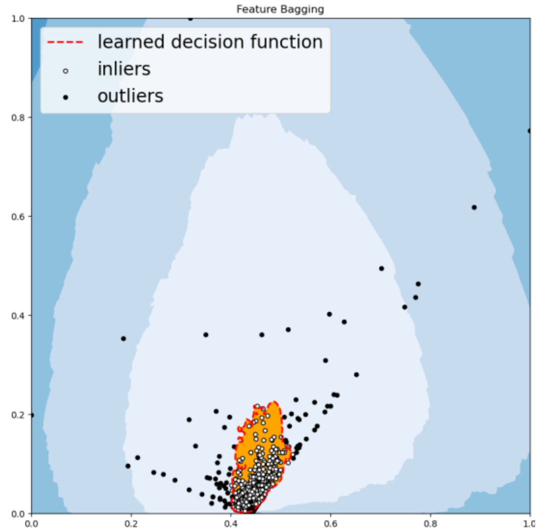
13. Experimentar diversos métodos de deteção de outliers a partir da library pyod

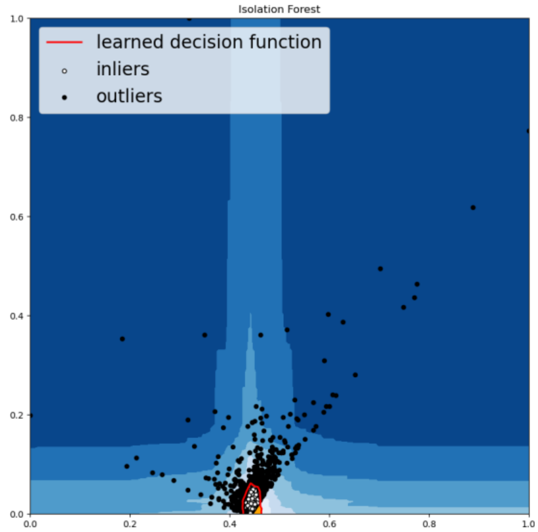
```
scaler = MinMaxScaler(feature_range=(0, 1))
df[['Profit', 'Sales']] = scaler.fit_transform(df[['Profit', 'Sales']])
X1 = df['Profit'].values.reshape(-1, 1)
X2 = df['Sales'].values.reshape(-1, 1)
X = np.concatenate((X1, X2), axis=1)

random_state = np.random.RandomState(42)
outliers_fraction = 0.05
# Define seven outlier detection tools to be compared
classifiers = {
    'Cluster-based Local Outlier Factor (CBLOF)': CBLOF(contamination=outliers_fraction,
        check_estimator=False, random_state=random_state),
    'Feature Bagging': FeatureBagging(LOF(n_neighbors=35),
       contamination=outliers_fraction, check_estimator=False,
       random_state=random_state),
    'Isolation Forest': IForest(contamination=outliers_fraction,
        random_state=random_state),
    'K Nearest Neighbors (KNN)': KNN(contamination=outliers_fraction),
    'Average KNN': KNN(method='mean', contamination=outliers_fraction)
}

xx, yy = np.meshgrid(np.linspace(0, 1, 200), np.linspace(0, 1, 200))
```

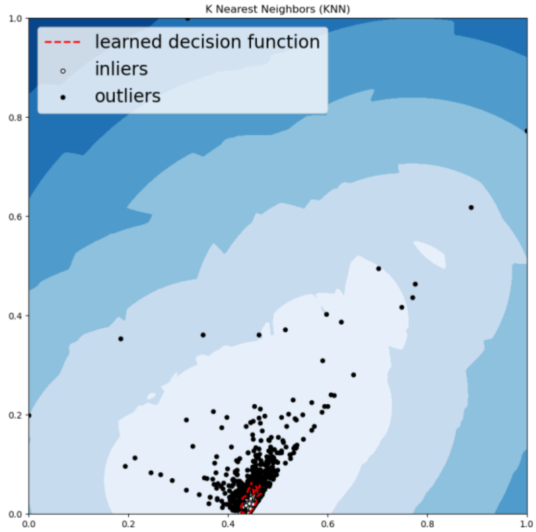
```
for i, (clf_name, clf) in enumerate(classifiers.items()):
    clf.fit(X)
    # predict raw anomaly score
    scores_pred = clf.decision_function(X) * -1

    # prediction of a datapoint category outlier or inlier
    y_pred = clf.predict(X)
    n_inliers = len(y_pred) - np.count_nonzero(y_pred)
    n_outliers = np.count_nonzero(y_pred == 1)
    plt.figure(figsize=(10, 10))

    # copy of dataframe
    dfx = df
    dfx['outlier'] = y_pred.tolist()

    # IX1 - inlier feature 1,  IX2 - inlier feature 2
    IX1 = np.array(dfx['Profit'][dfx['outlier'] == 0]).reshape(-1, 1)
    IX2 = np.array(dfx['Sales'][dfx['outlier'] == 0]).reshape(-1, 1)

    # OX1 - outlier feature 1, OX2 - outlier feature 2
    OX1 = dfx['Profit'][dfx['outlier'] == 1].values.reshape(-1, 1)
    OX2 = dfx['Sales'][dfx['outlier'] == 1].values.reshape(-1, 1)

    print('OUTLIERS : ', n_outliers, 'INLIERS : ', n_inliers, clf_name)

    # threshold value to consider a datapoint inlier or outlier
    threshold = stats.scoreatpercentile(scores_pred, 100 * outliers_fraction)

    # decision function calculates the raw anomaly score for every point
    Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()]) * -1
    Z = Z.reshape(xx.shape)

    # fill blue map colormap from minimum anomaly score to threshold value
    plt.contourf(xx, yy, Z, levels=np.linspace(Z.min(), threshold, 7),
         cmap=plt.cm.Blues_r)

    # draw red contour line where anomaly score is equal to thresold
    a = plt.contour(xx, yy, Z, levels=[threshold], linewidths=2, colors='red')

    # fill orange contour lines where range of anomaly score is from threshold
       to maximum anomaly score
    plt.contourf(xx, yy, Z, levels=[threshold, Z.max()], colors='orange')

    b = plt.scatter(IX1, IX2, c='white', s=20, edgecolor='k')

    c = plt.scatter(OX1, OX2, c='black', s=20, edgecolor='k')

    plt.axis('tight')

    # loc=2 is used for the top left corner
    plt.legend(
        [a.collections[0], b, c],
        ['learned decision function', 'inliers', 'outliers'],
        prop=matplotlib.font_manager.FontProperties(size=20),
        loc=2)

    plt.xlim((0, 1))
    plt.ylim((0, 1))
    plt.title(clf_name)
    plt.show()
```

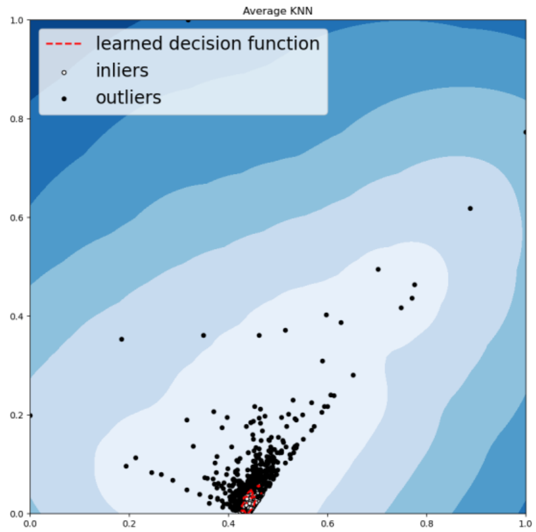
Resultados:


Cluster-based Local Outlier Factor (CBLOF)
learned decision function
inliers
outliers


Feature Bagging
learned decision function
inliers
outliers


Isolation Forest
learned decision function
inliers
outliers


K Nearest Neighbors (KNN)
learned decision function
inliers
outliers


Average KNN
learned decision function
inliers
outliers