This project analyzes real-world medical appointment data from hospitals in Brazil, focusing on why patients miss their scheduled appointments.

Dataset: 110k+ medical appointments Target Variable: No-show (whether a patient showed up or not) Features: Gender, Age, SMS reminders, Scheduled/Appointment dates, Neighbourhood, Scholarship program, etc. The main goal is to discover the factors influencing patient no-shows and visualize key insights from the data.

Key Insights

- Patients who received an SMS reminder were less likely to miss their appointments.
- Younger patients had higher no-show rates compared to older patients.
- Longer waiting times between scheduling and appointment increased no-show likelihood.

*Machine Learing descrebtion *

Absenteeism Prediction Model (Random Forest Classifier)

This project aims to predict employee absenteeism using a Random Forest Classifier.
The dataset includes features such as age, distance to work, waiting time for appointments, and gender.
The model is trained and optimized with GridSearchCV, and its performance is evaluated using confusion matrix and classification report.

Key Steps:

- Data cleaning and binary encoding (e.g., 'Gender', 'No-show')
- Train/test split using stratified sampling
- Model training and hyperparameter tuning
- Evaluation via precision, recall, F1-score, and confusion matrix
- Final visualization for interpretability

This model can help HR departments or healthcare clinics to identify individuals more likely to miss appointments or shifts, enabling targeted interventions.

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

```
# uploading the data
df= pd.read_csv('KaggleV2-May-2016.csv')
```

```
#abstarct of the data set
print(df.head())
print(df.shape)
```

```
       PatientId  AppointmentID Gender          ScheduledDay  \
0  2.987250e+13        5642903      F  2016-04-29T18:38:08Z
1  5.589980e+14        5642503      M  2016-04-29T16:08:27Z
2  4.262960e+12        5642549      F  2016-04-29T16:19:04Z
3  8.679510e+11        5642828      F  2016-04-29T17:29:31Z
4  8.841190e+12        5642494      F  2016-04-29T16:07:23Z

         AppointmentDay   Age      Neighbourhood  Scholarship  Hipertension  \
0  2016-04-29T00:00:00Z  62.0    JARDIM DA PENHA          0.0           1.0
1  2016-04-29T00:00:00Z  56.0    JARDIM DA PENHA          0.0           0.0
2  2016-04-29T00:00:00Z  62.0      MATA DA PRAIA          0.0           0.0
3  2016-04-29T00:00:00Z   8.0  PONTAL DE CAMBURI          0.0           0.0
4  2016-04-29T00:00:00Z  56.0    JARDIM DA PENHA          0.0           1.0

   Diabetes  Alcoholism  Handcap  SMS_received No-show
0       0.0         0.0      0.0           0.0      No
1       0.0         0.0      0.0           0.0      No
2       0.0         0.0      0.0           0.0      No
3       0.0         0.0      0.0           0.0      No
4       1.0         0.0      0.0           0.0      No
(32913, 14)
```

```
# data set info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32913 entries, 0 to 32912
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   PatientId       32913 non-null  float64
 1   AppointmentID   32913 non-null  int64
 2   Gender          32913 non-null  object
 3   ScheduledDay    32913 non-null  datetime64[ns, UTC]
 4   AppointmentDay  32912 non-null  datetime64[ns, UTC]
 5   Age             32912 non-null  float64
 6   Neighbourhood   32912 non-null  object
 7   Scholarship     32912 non-null  float64
 8   Hipertension    32912 non-null  float64
 9   Diabetes        32912 non-null  float64
 10  Alcoholism      32912 non-null  float64
 11  Handcap         32912 non-null  float64
 12  SMS_received    32912 non-null  float64
 13  No-show         32912 non-null  object
dtypes: datetime64[ns, UTC](2), float64(8), int64(1), object(3)
memory usage: 3.5+ MB
```

```
# describtion of data set
df.describe()
```

	PatientId	AppointmentID	Age	Scholarship	Hipertension	Diabetes	Alcoholism	Handcap	SMS_received
count	3.291300e+04	3.291300e+04	32912.000000	32912.000000	32912.000000	32912.000000	32912.000000	32912.000000	32912.000000
mean	1.494339e+14	5.656457e+06	36.830153	0.094829	0.192240	0.065478	0.041657	0.021937	0.310677
std	2.589637e+14	6.352510e+04	22.389541	0.292983	0.394067	0.247371	0.199806	0.159776	0.462778
min	9.380000e+04	5.030230e+06	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	4.193630e+12	5.631759e+06	18.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	3.112340e+13	5.664163e+06	37.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	9.441450e+13	5.698878e+06	54.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
max	9.999470e+14	5.754966e+06	98.000000	1.000000	1.000000	1.000000	1.000000	3.000000	1.000000

```
#duplicates, empties,...

print(df.isnull().sum())
print(df.duplicated().sum())
print(df['PatientId'].nunique(), '/', df['PatientId'].count())
```

```
PatientId         0
AppointmentID     0
Gender            0
ScheduledDay      0
AppointmentDay    0
Age               1
Neighbourhood     1
Scholarship       1
Hipertension      1
Diabetes          1
Alcoholism        1
Handcap           1
SMS_received      1
No-show           1
dtype: int64
0
22312 / 32913
```

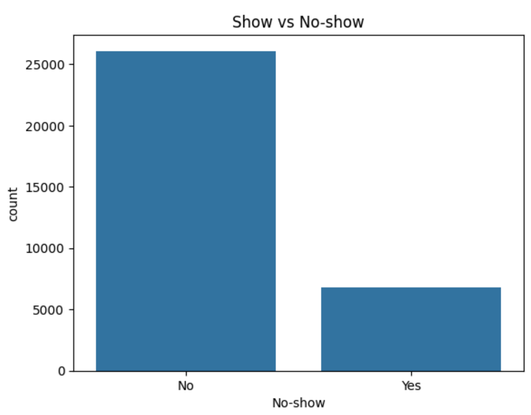
```
# changing into datetime

df['ScheduledDay'] = pd.to_datetime(df['ScheduledDay'])
df['AppointmentDay'] = pd.to_datetime(df['AppointmentDay'], errors='coerce')
```

```
# watting time/ new coulmn
df['WaitingDays'] = (df['AppointmentDay'] - df['ScheduledDay']).dt.days
```

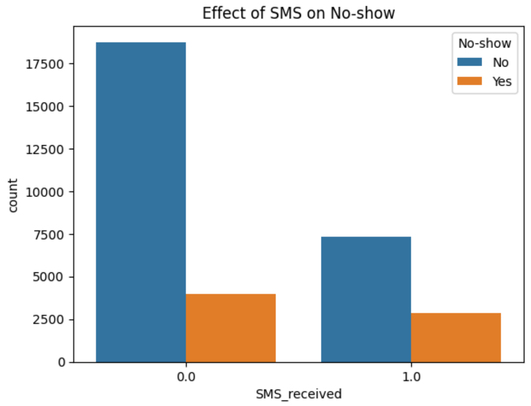
```
df = df[df['Age'] >= 0]
```

```
# visualizition
```

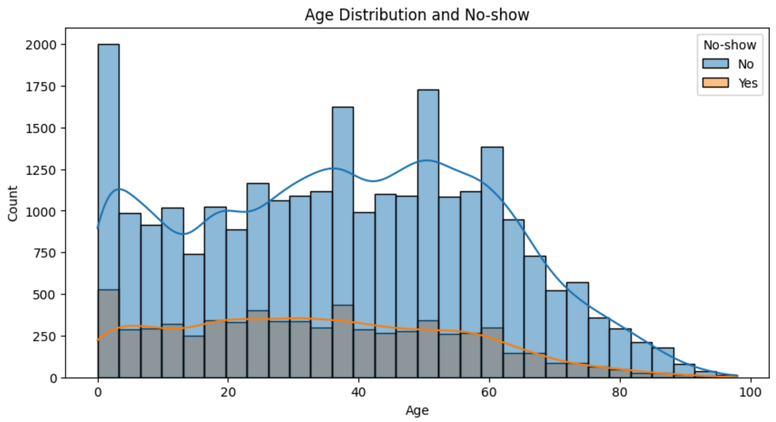
```
#show or no show
sns.countplot(data=df, x='No-show')
plt.title('Show vs No-show')
plt.show()
```



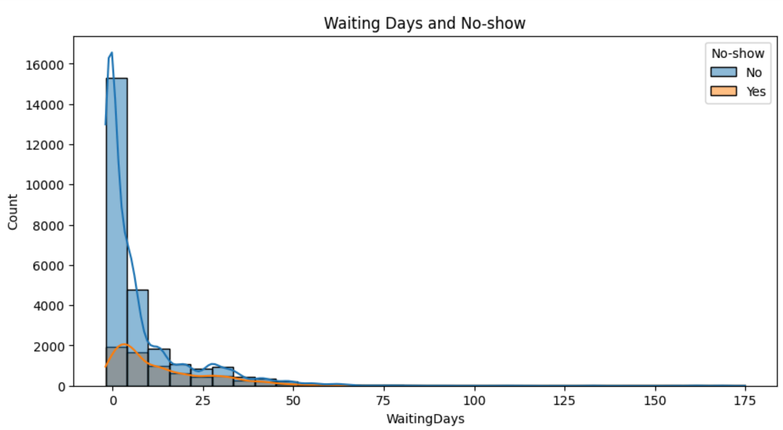

```
#impact of sms
sns.countplot(data=df, x='SMS_received', hue='No-show')
plt.title('Effect of SMS on No-show')
plt.show()
```




```
#histogram
plt.figure(figsize=(10,5))
sns.histplot(data=df, x='Age', hue='No-show', bins=30, kde=True)
plt.title('Age Distribution and No-show')
plt.show()
```




```
# watting time and no show
plt.figure(figsize=(10,5))
sns.histplot(data=df, x='WaitingDays', hue='No-show', bins=30, kde=True)
plt.title('Waiting Days and No-show')
plt.show()
```

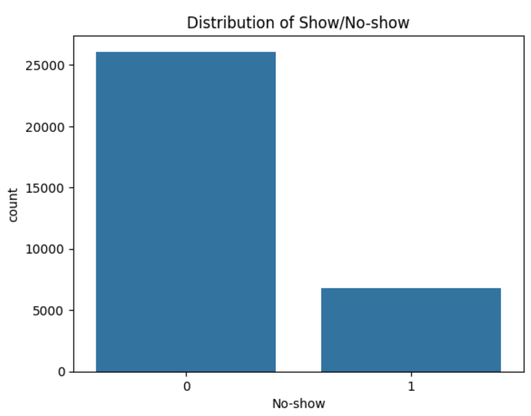



```
#No-show
df['No-show'] = df['No-show'].map({'No': 0, 'Yes': 1})
df['Gender'] = df['Gender'].map({'F': 0, 'M': 1})
```

```
/tmp/ipython-input-26-601432930.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['No-show'] = df['No-show'].map({'No': 0, 'Yes': 1})
/tmp/ipython-input-26-601432930.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

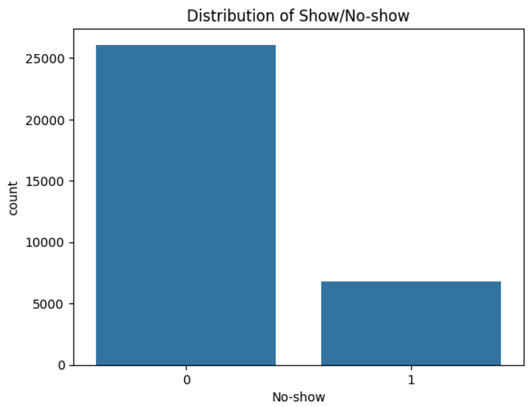
```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Gender'] = df['Gender'].map({'F': 0, 'M': 1})
```

```
sns.countplot(x='No-show', data=df)
plt.title("Distribution of Show/No-show")
plt.show()
```




```
print(df['No-show'].value_counts(normalize=True))

sns.countplot(x='No-show', data=df)
plt.title("Distribution of Show/No-show")
plt.show()
```

```
No-show
0    0.792477
1    0.207523
Name: proportion, dtype: float64
```




```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

```
print(df.columns)
```

```
Index(['PatientId', 'AppointmentID', 'Gender', 'ScheduledDay',
       'AppointmentDay', 'Age', 'Neighbourhood', 'Scholarship', 'Hipertension',
       'Diabetes', 'Alcoholism', 'Handcap', 'SMS_received', 'No-show',
       'WaitingDays'],
      dtype='object')
```

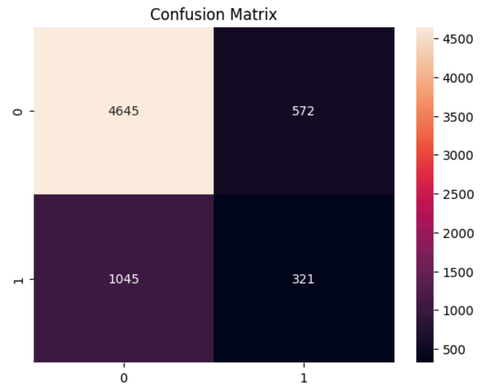
```
features = ['Age', 'Gender', 'Scholarship', 'Hipertension', 'Diabetes',
            'Alcoholism', 'Handcap', 'SMS_received', 'WaitingDays']

X = df[features]
y = df['No-show']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, random_state=42)

#model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
y_pred = model.predict(X_test)

print(classification_report(y_test, y_pred))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d')
plt.title("Confusion Matrix")
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.82      0.89      0.85      5217
           1       0.36      0.23      0.28      1366

    accuracy                           0.75      6583
   macro avg       0.59      0.56      0.57      6583
weighted avg       0.72      0.75      0.73      6583
```



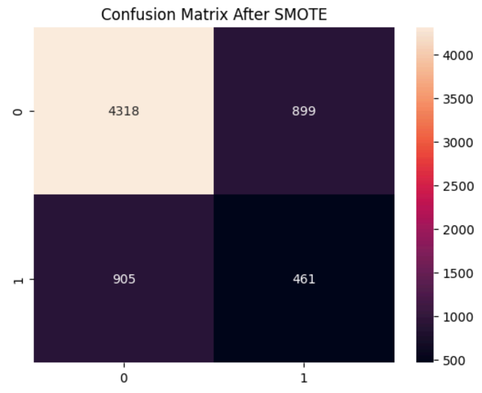

```
model = RandomForestClassifier(class_weight='balanced', random_state=42)

from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
```

```
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=42)
X_resampled, y_resampled = sm.fit_resample(X_train, y_train)

model = RandomForestClassifier(random_state=42)
model.fit(X_resampled, y_resampled)
```

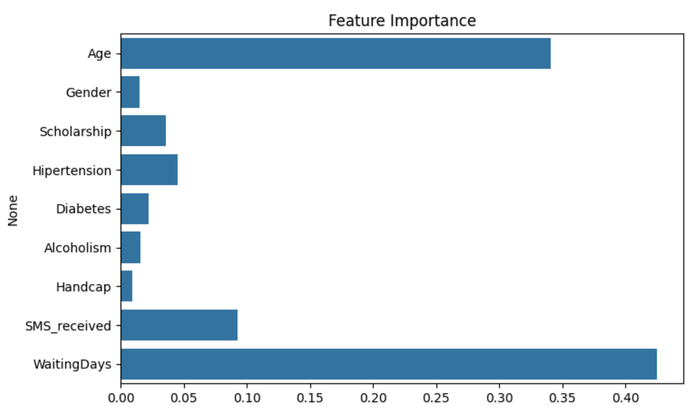
```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
from sklearn.metrics import classification_report, confusion_matrix
y_pred = model.predict(X_test)

print(classification_report(y_test, y_pred))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d')
plt.title("Confusion Matrix After SMOTE")
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.83      0.83      0.83      5217
           1       0.34      0.34      0.34      1366

    accuracy                           0.73      6583
   macro avg       0.58      0.58      0.58      6583
weighted avg       0.73      0.73      0.73      6583
```



```
importances = model.feature_importances_
feat_names = X.columns

plt.figure(figsize=(8,5))
sns.barplot(x=importances, y=feat_names)
plt.title("Feature Importance")
plt.show()
```




```
from sklearn.metrics import roc_auc_score

y_pred_proba = model.predict_proba(X_test)[:, 1]
roc_auc_score(y_test, y_pred_proba)
```

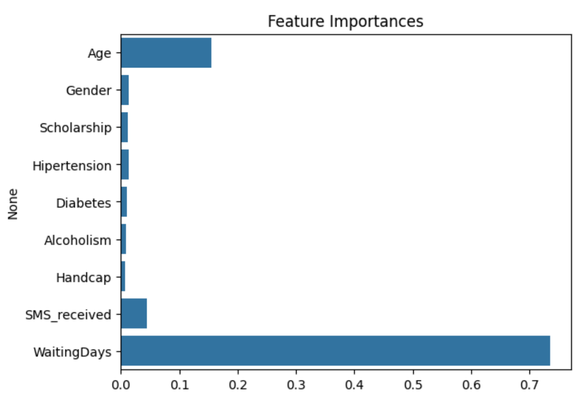
```
np.float64(0.6565047649437543)
```

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(class_weight='balanced', random_state=42)
model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(class_weight='balanced', random_state=42)
```

```
from sklearn.model_selection import GridSearchCV

params = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
}

grid = GridSearchCV(RandomForestClassifier(class_weight='balanced', random_state=42),
                    param_grid=params,
                    scoring='recall',
                    cv=3,
                    n_jobs=-1)
grid.fit(X_train, y_train)

best_model = grid.best_estimator_
```

```
importances = best_model.feature_importances_
feat_names = X.columns
sns.barplot(x=importances, y=feat_names)
plt.title("Feature Importances")
plt.show()
```



```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

grid_search = GridSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_grid=param_grid,
    cv=3,
    scoring='f1',
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X_resampled, y_resampled)

best_model = grid_search.best_estimator_


y_pred = best_model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
              precision    recall  f1-score   support

           0       0.83      0.83      0.83      5217
           1       0.34      0.34      0.34      1366

    accuracy                           0.73      6583
   macro avg       0.58      0.58      0.58      6583
weighted avg       0.73      0.73      0.73      6583
```

```
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)
```

```
Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Best Score: 0.7772052798768622
```

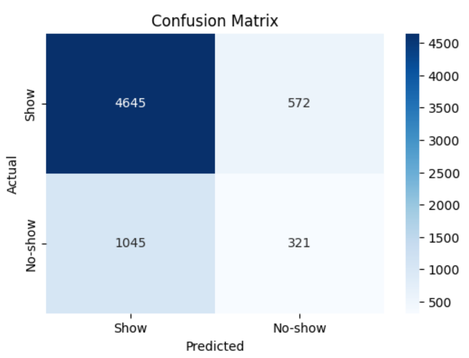
```
best_model = grid_search.best_estimator_
best_model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

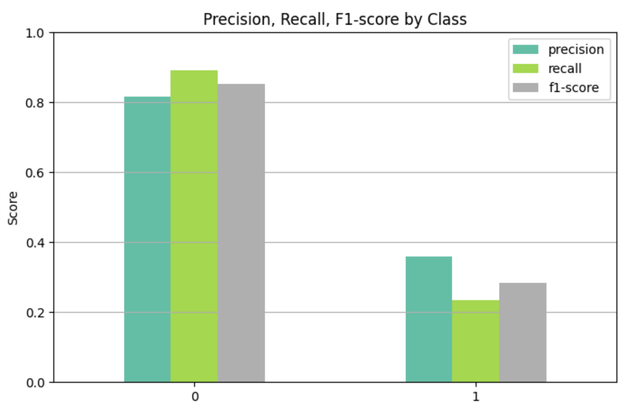
```
from sklearn.metrics import classification_report, confusion_matrix

y_pred = best_model.predict(X_test)

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[4645  572]
 [1045  321]]

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.89      0.85      5217
           1       0.36      0.23      0.28      1366

    accuracy                           0.75      6583
   macro avg       0.59      0.56      0.57      6583
weighted avg       0.72      0.75      0.73      6583
```

```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Show', 'No-show'], yticklabels=['Show', 'No-show'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



```
report_dict = classification_report(y_test, y_pred, output_dict=True)
df_report = pd.DataFrame(report_dict).transpose()

df_report.iloc[:2, :3].plot(kind='bar', figsize=(8,5), colormap='Set2')
plt.title('Precision, Recall, F1-score by Class')
plt.xticks(rotation=0)
plt.ylabel('Score')
plt.ylim(0, 1)
plt.grid(axis='y')
plt.show()
```




```
#example
sample = X_test.iloc[[10]]
real = y_test.iloc[10]
pred = best_model.predict(sample)[0]

print("ویژگی‌ها:", sample.to_dict())
print(f"پیش‌بینی مدل: {'No-show' if pred==1 else 'Show'}")
print(f"وضعیت واقعی: {'No-show' if real==1 else 'Show'}")
```

```
ویژگی‌ها: {'Age': {29596: 39.0}, 'Gender': {29596: 0}, 'Scholarship': {29596: 0.0}, 'Hipertension': {29596: 1.0}, 'Diabetes': {29596: 0.0}, 'Alcoholism': {29596: 0.0}, 'Handcap': {29596: 0.0}, 'SMS_received': {29596: 1.0}, 'WaitingDays': {29596: 5.0}}
پیش‌بینی مدل: Show
وضعیت واقعی: Show
```