Data Analysis Health

The client has provided one year's worth of timeline raw data for one of their facilities. The team will need you to clean it up and analyze it. The data shows the timeline for around 5,000 patient visits, and includes the following information about the visits:

```
What hour of the day the patient arrived
How long check-in takes
How long the patient has to wait before they meet with a nurse to get vitals
taken (blood pressure, heart rate, etc.)
How long it takes the nurse to complete the vitals examination
How long the patient has to wait between getting vitals taken and meeting with
a doctor
How long the patient is with the doctor
How long it takes the patient to complete the final payment/insurance
Which receptionist each patient works with
Which diagnosis group the doctor has tagged them (Group 1 = most severe; Group
3 = least severe)
```

Import Libraries

In [1]:
```
import numpy as np
from numpy import count_nonzero, median, mean
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random

# Plotly
# import plotly.express as px
# import plotly.offline as py
# import plotly.graph_objs as go

import sweetviz

import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.formula.api import ols
# import researchpy as rp

import datetime
from datetime import datetime, timedelta

# import eli5
# from IPython.display import display

#import os
#import zipfile
import scipy.stats
from collections import Counter

import sklearn
# from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder, OneHotEn
# from sklearn.linear_model import LinearRegression, LogisticRegression, ElasticNet, Las
# from sklearn.model_selection import cross_val_score, train_test_split
```

```
# from sklearn.metrics import accuracy_score, auc, classification_report, confusion_matr
# from sklearn.metrics import plot_confusion_matrix, plot_roc_curve

# from sklearn.linear_model import ElasticNet, Lasso, LinearRegression, LogisticRegressi
# from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, ExtraTreeClass
# from sklearn.svm import SVC, SVR, LinearSVC, LinearSVR
# from sklearn.naive_bayes import GaussianNB, MultinomialNB

%matplotlib inline
#sets the default autosave frequency in seconds
%autosave 60
sns.set_style('dark')
sns.set(font_scale=1.2)

plt.rc('axes', titlesize=9)
plt.rc('axes', labelsize=14)
plt.rc('xtick', labelsize=12)
plt.rc('ytick', labelsize=12)

import warnings
warnings.filterwarnings('ignore')

# Use Feature-Engine library
#import feature_engine
#from feature_engine import imputation as mdi
#from feature_engine.outlier_removers import Winsorizer
#from feature_engine import categorical_encoders as ce
#from feature_engine.discretisation import EqualWidthDiscretiser, EqualFrequencyDiscreti
#from feature_engine.discretisation import ArbitraryDiscretiser, DecisionTreeDiscretiser
#from feature_engine.encoding import OrdinalEncoder

pd.set_option('display.max_columns',None)
#pd.set_option('display.max_rows',None)
pd.set_option('display.width', 1000)
pd.set_option('display.float_format','{:.2f}'.format)

random.seed(0)
np.random.seed(0)
np.set_printoptions(suppress=True)
```

```
Autosaving every 60 seconds
```

Exploratory Data Analysis

In [2]:
```
df = pd.read_csv("Healthy.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	Unique ID	Year	Arrival(Hour of day, GMT)	Check-in(mins)	Waitforvitals(mins)	Vitals(mins)	Waitfordoctor(mins)	Doctorvisit(mins)
0	476	2019	11	1.00	12.00	3.00	13.00	8.10
1	902	2019	20	2.00	10.00	5.75	13.00	7.20
2	1236	2019	9	3.00	24.70	4.60	23.40	6.00
3	1925	2019	11	2.00	14.00	5.75	18.00	3.60
4	1999	2019	13	3.00	16.50	4.20	22.00	7.20

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5007 entries, 0 to 5006
Data columns (total 13 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Unique ID                   5007 non-null   int64  
 1   Year                        5007 non-null   int64  
 2   Arrival(Hour of day, GMT)   5007 non-null   int64  
 3   Check-in(mins)              5007 non-null   float64
 4   Waitforvitals(mins)         5007 non-null   float64
 5   Vitals(mins)                5007 non-null   float64
 6   Waitfordoctor(mins)         5007 non-null   float64
 7   Doctorvisit(mins)           5007 non-null   float64
 8   Payment(mins)               5007 non-null   float64
 9   Diagnosisgroup              5007 non-null   object 
 10  Admin(check-in)             5007 non-null   object 
 11  Nurse(vitals)               5007 non-null   object 
 12  Doctor                      5007 non-null   object 
dtypes: float64(6), int64(3), object(4)
memory usage: 508.6+ KB
```

In [5]:
```
df.describe(include='all')
```

Out[5]:

	Unique ID	Year	Arrival(Hour of day, GMT)	Check-in(mins)	Waitforvitals(mins)	Vitals(mins)	Waitfordoctor(mins)	Doctorv
count	5007.00	5007.00	5007.00	5007.00	5007.00	5007.00	5007.00	
unique	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
top	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
freq	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
mean	503073.31	2019.00	14.50	2.79	16.39	5.12	23.30	
std	290148.82	0.00	3.47	1.45	6.47	1.42	8.12	
min	476.00	2019.00	9.00	1.00	2.55	2.55	6.80	
25%	257003.50	2019.00	11.00	2.00	12.10	4.20	18.00	
50%	504724.00	2019.00	15.00	2.00	15.40	4.75	22.00	
75%	753581.00	2019.00	18.00	4.00	20.00	5.75	27.50	
max	999924.00	2019.00	20.00	5.75	43.75	9.20	76.05	

In [6]:
```
df.columns
```

Out[6]:
```
Index(['Unique ID', 'Year', 'Arrival(Hour of day, GMT)', 'Check-in(mins)', 'Waitforvitals(mins)', 'Vitals(mins)', 'Waitfordoctor(mins)', 'Doctorvisit(mins)', 'Payment(mins)', 'Diagnosisgroup', 'Admin(check-in)', 'Nurse(vitals)', 'Doctor'], dtype='object')
```

In [7]:
```
df.Year.value_counts()
```

Out[7]:
```
2019    5007
Name: Year, dtype: int64
```

In [8]:
```
df["Admin(check-in)"].value_counts()
```

Out[8]:
```
Smith, J    2509
Akapo, K    1871
Ramos, T     627
Name: Admin(check-in), dtype: int64
```

In [9]:
```
df["Nurse(vitals)"].value_counts()
```

Out[9]:
```
Kumar, S       883
Lumantas, M    731
Birdsall, A    640
Aros, W        625
Knight, F      580
Turner, P      552
Musk, J        532
Banu, H        464
Name: Nurse(vitals), dtype: int64
```

In [10]:
```
df["Doctor"].value_counts()
```

Out[10]:
```
Dr. Jankowski    1145
Dr. Yung         1007
Dr. Balla        1000
Dr. Campbell      996
Dr. Campos        859
Name: Doctor, dtype: int64
```

Groupby Function

In [11]:
```
df.groupby(["Doctor"])["Check-in(mins)",'Waitforvitals(mins)', 'Vitals(mins)', 'Waitford
```

Out[11]:

Doctor	Check-in(mins)	Waitforvitals(mins)	Vitals(mins)	Waitfordoctor(mins)	Doctorvisit(mins)	Payment(mins)
Dr. Balla	2.84	16.51	5.15	29.64	6.79	3.61
Dr. Campbell	2.75	16.27	5.17	21.85	6.43	3.65
Dr. Campos	2.76	16.56	5.03	21.41	6.33	3.62
Dr. Jankowski	2.74	16.34	5.12	21.83	6.42	3.54
Dr. Yung	2.84	16.31	5.12	21.71	6.27	3.50

In [12]:
```
df.groupby(["Doctor"])["Check-in(mins)",'Waitforvitals(mins)', 'Vitals(mins)', 'Waitford
```

Out[12]:

Doctor	Check-in(mins)	Waitforvitals(mins)	Vitals(mins)	Waitfordoctor(mins)	Doctorvisit(mins)	Payment(mins)
Dr. Balla	1000	1000	1000	1000	1000	1000
Dr. Campbell	996	996	996	996	996	996
Dr. Campos	859	859	859	859	859	859
Dr. Jankowski	1145	1145	1145	1145	1145	1145

Dr. Yung	1007	1007	1007	1007	1007	1007

In [13]:
```
df.groupby(["Diagnosisgroup"])["Check-in(mins)",'Waitforvitals(mins)', 'Vitals(mins)', '
```

Out[13]:

Diagnosisgroup	Check-in(mins)	Waitforvitals(mins)	Vitals(mins)	Waitfordoctor(mins)	Doctorvisit(mins)	Payment(mins)
Group 1	2.85	16.35	5.07	22.83	11.18	3.42
Group 2	2.83	16.47	5.16	23.57	6.76	3.60
Group 3	2.75	16.35	5.12	23.26	5.33	3.61

In [14]:
```
df.groupby(["Nurse(vitals)"])["Check-in(mins)",'Waitforvitals(mins)', 'Vitals(mins)', 'W
```

Out[14]:

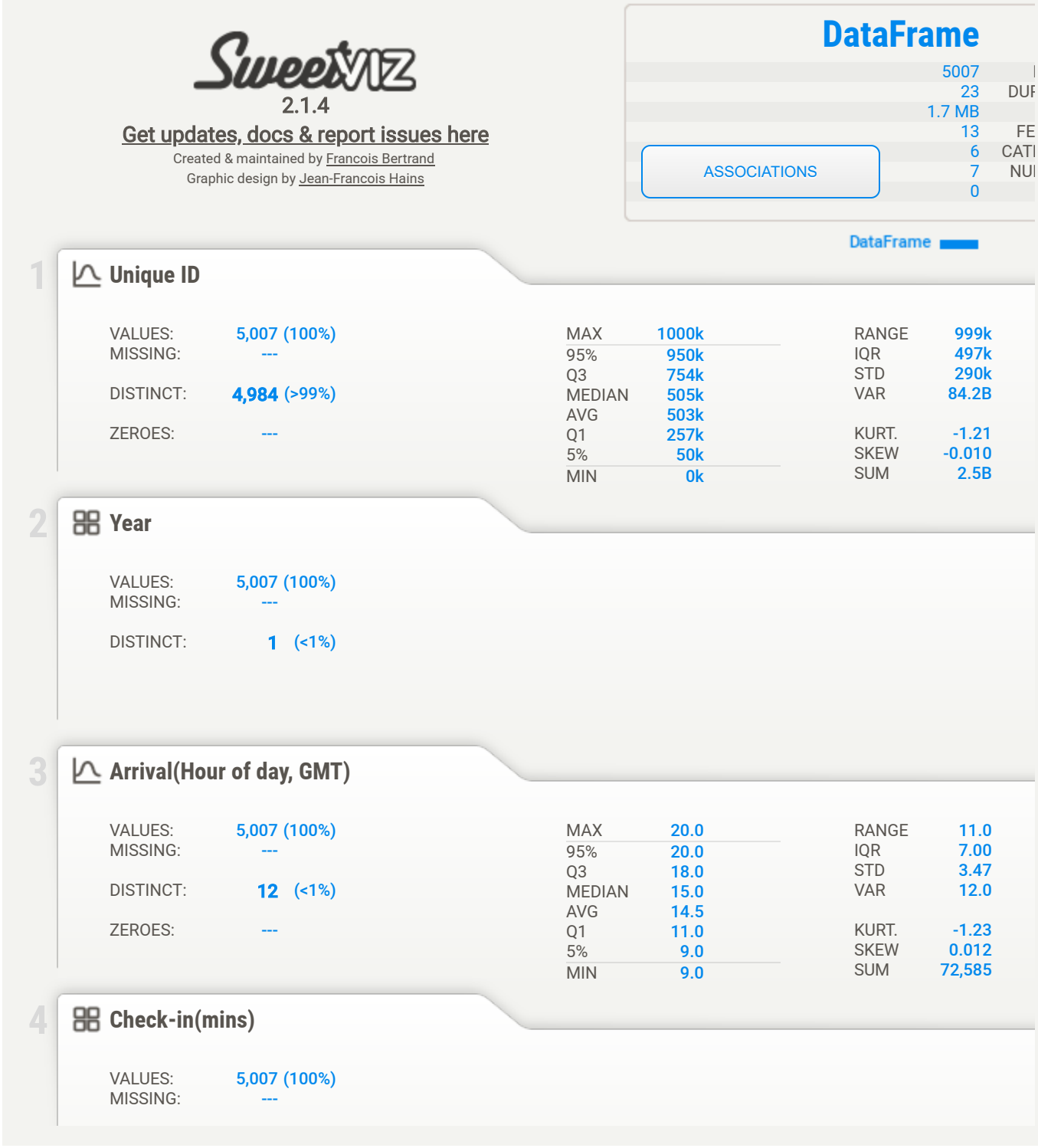
Nurse(vitals)	Check-in(mins)	Waitforvitals(mins)	Vitals(mins)	Waitfordoctor(mins)	Doctorvisit(mins)	Payment(mins)
Aros, W	2.72	16.41	4.69	23.49	6.21	3.49
Banu, H	2.75	16.18	5.67	23.07	6.41	3.59
Birdsall, A	2.82	16.27	4.18	23.08	6.43	3.53
Knight, F	2.86	16.61	5.45	23.04	6.47	3.72
Kumar, S	2.77	16.13	4.96	23.22	6.60	3.59
Lumantas, M	2.78	16.51	5.14	23.39	6.40	3.70
Musk, J	2.78	16.51	5.64	23.47	6.64	3.61
Turner, P	2.82	16.59	5.66	23.61	6.37	3.40

SweetViz Reports

In [15]:
```
report = sweetviz.analyze(df)
report.show_html("analysis.html")
```

```
                                             |
   | [  0%]   00:00 ->…
Report analysis.html was generated! NOTEBOOK/COLAB USERS: the web browser MAY not pop u
p, regardless, the report IS saved in your notebook/colab files.
```

In [16]:
```
report.show_notebook()
```

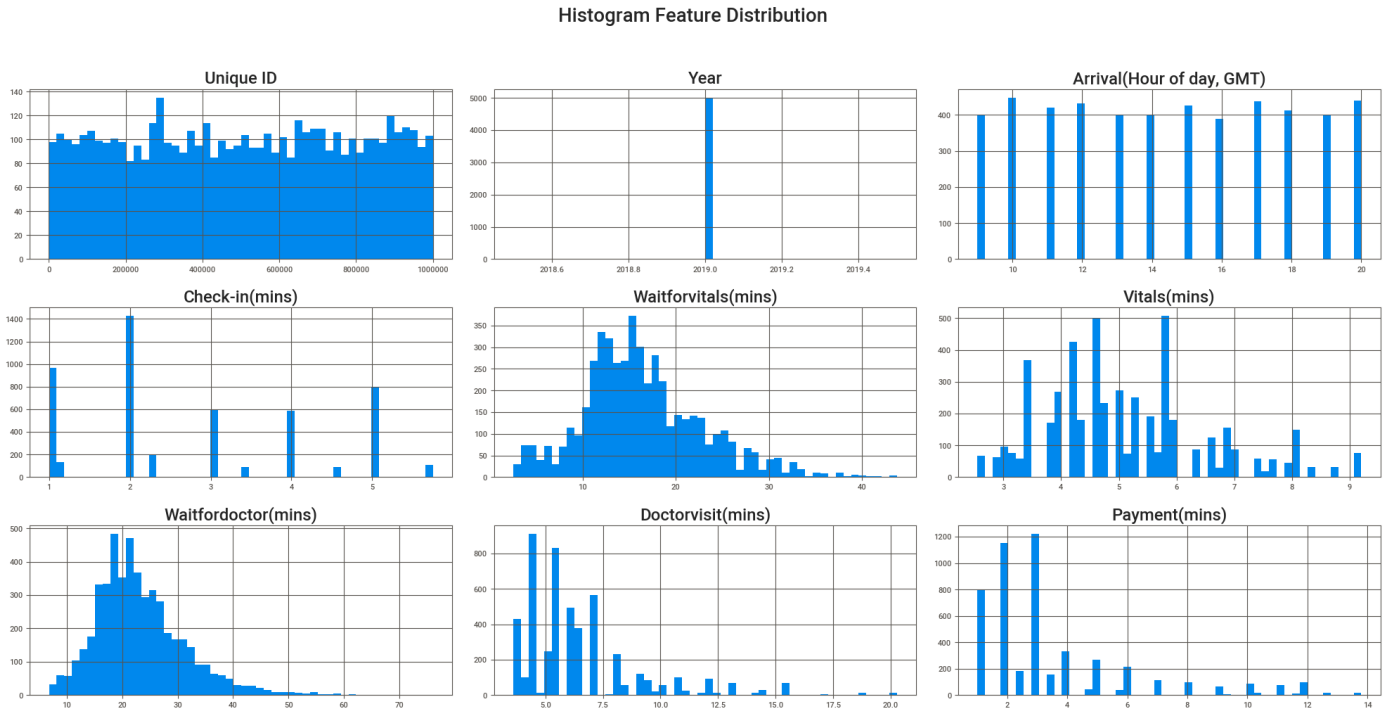


Data Visualization

Univariate Data Exploration

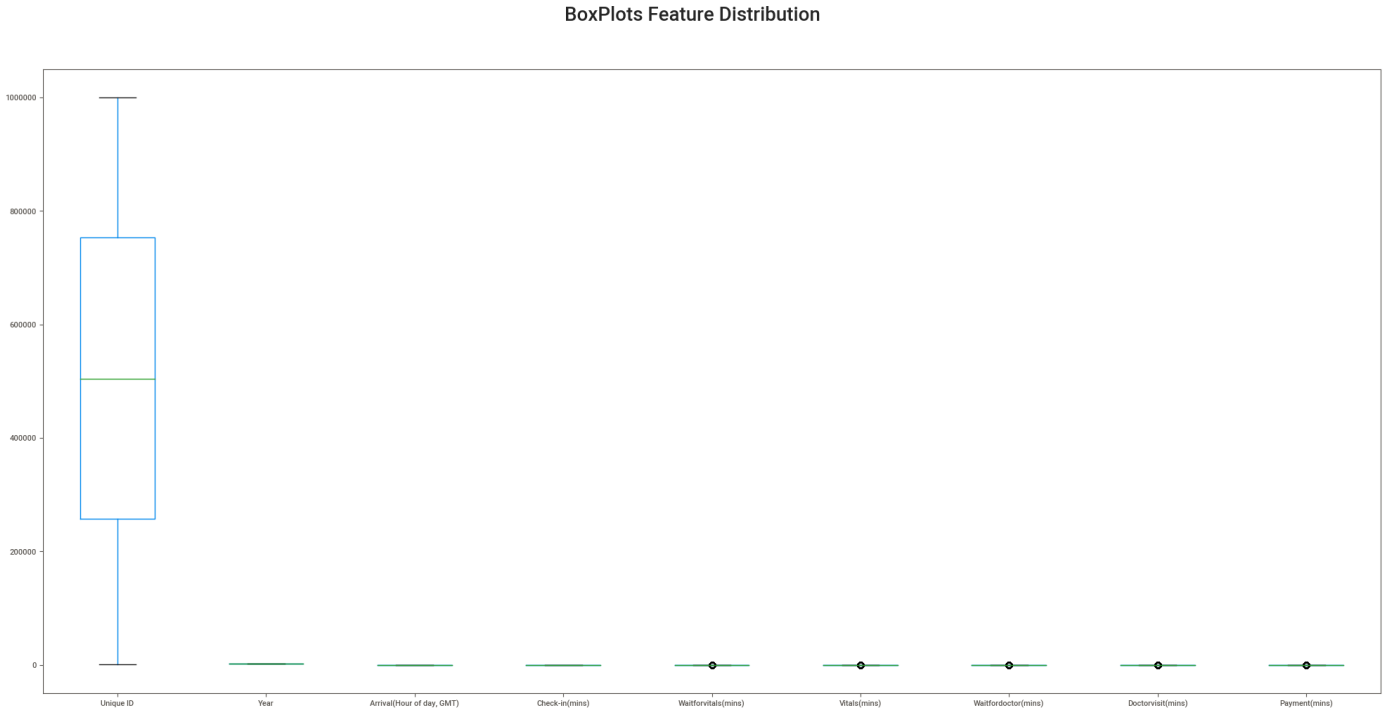
In [17]:
```
df.hist(bins=50, figsize=(20,10))
plt.suptitle('Histogram Feature Distribution', x=0.5, y=1.02, ha='center', fontsize=20)

plt.tight_layout()
plt.show()
```



In [18]:
```
df.boxplot(figsize=(20,10), grid=False)
plt.suptitle('BoxPlots Feature Distribution', x=0.5, y=1.02, ha='center', fontsize=20)

plt.tight_layout()
plt.show()
```



Data Preprocessing

Feature Engineering

In [19]:
```
df.head()
```

Out[19]:

Unique ID	Year	Arrival(Hour of day, GMT)	Check-in(mins)	Waitforvitals(mins)	Vitals(mins)	Waitfordoctor(mins)	Doctorvisit(mins)

0	476	2019	11	1.00	12.00	3.00	13.00	8.10
1	902	2019	20	2.00	10.00	5.75	13.00	7.20
2	1236	2019	9	3.00	24.70	4.60	23.40	6.00
3	1925	2019	11	2.00	14.00	5.75	18.00	3.60
4	1999	2019	13	3.00	16.50	4.20	22.00	7.20

In [20]:
```
df["TotalWaitTimes"] = df["Waitforvitals(mins)"] + df["Vitals(mins)"]
```

In [21]:
```
df["TotalDoctorTimes"] = df["Waitfordoctor(mins)"] + df["Doctorvisit(mins)"]
```

In [22]:
```
df.head(1)
```

Out[22]:

	Unique ID	Year	Arrival(Hour of day, GMT)	Check-in(mins)	Waitforvitals(mins)	Vitals(mins)	Waitfordoctor(mins)	Doctorvisit(mins)
0	476	2019	11	1.00	12.00	3.00	13.00	8.10

Treat Duplicate Values

In [23]:
```
df.duplicated(keep='first').sum()
```

Out[23]:
```
23
```

In [24]:
```
df[df.duplicated(keep=False)] #Check duplicate values
```

Out[24]:

	Unique ID	Year	Arrival(Hour of day, GMT)	Check-in(mins)	Waitforvitals(mins)	Vitals(mins)	Waitfordoctor(mins)	Doctorvisit(mir
182	36884	2019	19	1.00	12.50	5.50	43.75	6.
183	36884	2019	19	1.00	12.50	5.50	43.75	6.
402	81658	2019	13	5.00	14.30	4.00	23.10	10.
403	81658	2019	13	5.00	14.30	4.00	23.10	10.
432	87029	2019	16	1.00	15.30	5.00	17.00	5.
433	87029	2019	16	1.00	15.30	5.00	17.00	5.
1068	216097	2019	20	4.00	12.00	7.70	22.00	6.
1069	216097	2019	20	4.00	12.00	7.70	22.00	6.
1599	318105	2019	11	1.00	14.00	5.10	27.00	8.
1600	318105	2019	11	1.00	14.00	5.10	27.00	8.

1646	328226	2019	15	2.00	16.20	3.40	16.20	7.
1647	328226	2019	15	2.00	16.20	3.40	16.20	7.
2027	405194	2019	9	2.00	18.20	8.00	19.50	5.
2028	405194	2019	9	2.00	18.20	8.00	19.50	5.
2347	472455	2019	12	2.00	26.25	4.00	37.05	6.
2348	472455	2019	12	2.00	26.25	4.00	37.05	6.
3077	622164	2019	19	5.75	10.00	2.85	37.50	5.
3078	622164	2019	19	5.75	10.00	2.85	37.50	5.
3206	648949	2019	14	3.00	10.45	4.60	30.88	6.
3207	648949	2019	14	3.00	10.45	4.60	30.88	6.
3280	663171	2019	11	2.00	13.00	5.25	10.00	6.
3281	663171	2019	11	2.00	13.00	5.25	10.00	6.
3316	669778	2019	11	2.00	10.00	7.70	26.52	7.
3317	669778	2019	11	2.00	10.00	7.70	26.52	7.
3341	674261	2019	15	4.60	15.30	4.25	22.50	8.
3342	674261	2019	15	4.60	15.30	4.25	22.50	8.
4148	836263	2019	11	3.45	20.00	3.45	25.00	5.
4149	836263	2019	11	3.45	20.00	3.45	25.00	5.
4641	929491	2019	10	3.00	27.50	5.50	22.00	6.
4642	929491	2019	10	3.00	27.50	5.50	22.00	6.
4921	983613	2019	12	2.00	16.25	9.20	30.00	6.
4922	983613	2019	12	2.00	16.25	9.20	30.00	6.
4923	983613	2019	12	2.00	16.25	9.20	30.00	6.
4924	983613	2019	12	2.00	16.25	9.20	30.00	6.
4925	983613	2019	12	2.00	16.25	9.20	30.00	6.
4926	983613	2019	12	2.00	16.25	9.20	30.00	6.
4927	983613	2019	12	2.00	16.25	9.20	30.00	6.
4928	983613	2019	12	2.00	16.25	9.20	30.00	6.
4929	983613	2019	12	2.00	16.25	9.20	30.00	6.

In [25]:
```
df.drop_duplicates(ignore_index=True, inplace=True)
```

In [26]:
```
df
```

Out[26]:

	Unique ID	Year	Arrival(Hour of day, GMT)	Check-in(mins)	Waitforvitals(mins)	Vitals(mins)	Waitfordoctor(mins)	Doctorvisit(mir
0	476	2019	11	1.00	12.00	3.00	13.00	8.
1	902	2019	20	2.00	10.00	5.75	13.00	7.
2	1236	2019	9	3.00	24.70	4.60	23.40	6.
3	1925	2019	11	2.00	14.00	5.75	18.00	3.
4	1999	2019	13	3.00	16.50	4.20	22.00	7.
...	...	...	...	...	...	...	...	
4979	998699	2019	10	5.00	27.50	3.80	16.50	6.
4980	998705	2019	17	5.00	14.25	4.00	20.90	3.
4981	999064	2019	15	4.00	12.60	4.60	22.23	3.
4982	999605	2019	20	2.30	17.00	3.45	20.00	3.
4983	999924	2019	19	3.00	41.25	7.00	33.75	7.

4984 rows × 15 columns

In [27]:
```
df.reset_index(drop=True, inplace=True)
```

Remove columns

In [28]:
```
df.columns
```

Out[28]:
```
Index(['Unique ID', 'Year', 'Arrival(Hour of day, GMT)', 'Check-in(mins)', 'Waitforvitals(mins)', 'Vitals(mins)', 'Waitfordoctor(mins)', 'Doctorvisit(mins)', 'Payment(mins)', 'Diagnosisgroup', 'Admin(check-in)', 'Nurse(vitals)', 'Doctor', 'TotalWaitTimes', 'TotalDoctorTimes'], dtype='object')
```

In [29]:
```
df.drop(['Unique ID', 'Year', 'Admin(check-in)', 'Nurse(vitals)'], axis=1, inplace=True)
```

In [30]:
```
df
```

Out[30]:

	Arrival(Hour of day, GMT)	Check-in(mins)	Waitforvitals(mins)	Vitals(mins)	Waitfordoctor(mins)	Doctorvisit(mins)	Payment(m
0	11	1.00	12.00	3.00	13.00	8.10	
1	20	2.00	10.00	5.75	13.00	7.20	
2	9	3.00	24.70	4.60	23.40	6.00	
3	11	2.00	14.00	5.75	18.00	3.60	

4	13	3.00	16.50	4.20	22.00	7.20	
...	...	...	...	...	...	...	
4979	10	5.00	27.50	3.80	16.50	6.00	
4980	17	5.00	14.25	4.00	20.90	3.60	
4981	15	4.00	12.60	4.60	22.23	3.60	
4982	20	2.30	17.00	3.45	20.00	3.60	
4983	19	3.00	41.25	7.00	33.75	7.20	

4984 rows × 11 columns

In [ ]:

One-hot encoding

In [31]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4984 entries, 0 to 4983
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Arrival(Hour of day, GMT)   4984 non-null   int64  
 1   Check-in(mins)              4984 non-null   float64
 2   Waitforvitals(mins)         4984 non-null   float64
 3   Vitals(mins)                4984 non-null   float64
 4   Waitfordoctor(mins)         4984 non-null   float64
 5   Doctorvisit(mins)           4984 non-null   float64
 6   Payment(mins)               4984 non-null   float64
 7   Diagnosisgroup              4984 non-null   object 
 8   Doctor                      4984 non-null   object 
 9   TotalWaitTimes              4984 non-null   float64
 10  TotalDoctorTimes            4984 non-null   float64
dtypes: float64(8), int64(1), object(2)
memory usage: 428.4+ KB
```

In [32]:
```
df2 = pd.get_dummies(data=df,drop_first=True)
```

In [33]:
```
df2
```

Out[33]:

	Arrival(Hour of day, GMT)	Check-in(mins)	Waitforvitals(mins)	Vitals(mins)	Waitfordoctor(mins)	Doctorvisit(mins)	Payment(m
0	11	1.00	12.00	3.00	13.00	8.10	
1	20	2.00	10.00	5.75	13.00	7.20	
2	9	3.00	24.70	4.60	23.40	6.00	
3	11	2.00	14.00	5.75	18.00	3.60	
4	13	3.00	16.50	4.20	22.00	7.20	
...	...	...	...	...	...	...	
4979	10	5.00	27.50	3.80	16.50	6.00	

4980	17	5.00	14.25	4.00	20.90	3.60	
4981	15	4.00	12.60	4.60	22.23	3.60	
4982	20	2.30	17.00	3.45	20.00	3.60	
4983	19	3.00	41.25	7.00	33.75	7.20	

4984 rows × 15 columns

Save to CSV

In [34]:
```
df.to_csv("healthrevised.csv", index=False)
```

Regression Analysis

Linear Regression (StatsModel)

In [ ]:
```
df.columns
```

In [ ]:
```
y = df['ExpirationMonth']
X = df['NumStores']
```

In [ ]:
```
X = sm.add_constant(X)
```

In [ ]:
```
model = sm.OLS(y,X).fit()
```

In [ ]:
```
model.summary()
```

In [ ]:
```
prediction = model.predict(x)
```

In [ ]:
```

```

In [ ]:
```
linreg = smf.ols(formula='Lottery ~ Literacy + Wealth + Region', data=df).fit()
```

Residual Plots

In [ ]:
```
fig = plt.figure(figsize=(12,8))
fig = sm.graphics.plot_regress_exog(model, 'x_variables', fig=fig)
```

In [ ]:
```
fig = plt.figure(figsize=(12,8))
fig = sm.graphics.plot_ccpr(prestige_model, "education")
fig.tight_layout(pad=1.0)
```

Linear Regression (SKLearn)

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

Logistic Regression (StatsModel)

In [ ]:
```
df.columns
```

In [ ]:
```
y = df['']
X = df['']
```

In [ ]:
```
X = sm.add_constant(X)
```

In [ ]:
```
model = sm.Logit(y, X).fit()
```

In [ ]:
```
model.summary()
```

In [ ]:
```
logitfit = smf.logit(formula = 'DF ~ Debt_Service_Coverage + cash_security_to_curLiab +
```

In [ ]:
```
logitfit = smf.logit(formula = 'DF ~ TNW + C(seg2)', data = hgcdev).fit()
```

Logistic Regression (SKLearn)

In [ ]:
```
df.shape
```

In [ ]:
```
X = df.iloc[:,:4]
y = df.iloc[:,4]
```

In [ ]:
```
Counter(y)
```

In [ ]:
```
X.values, y.values
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X.values, y.values, test_size=0.2, r
```

In [ ]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

In [ ]:
```
Counter(y_train), Counter(y_test)
```

In [ ]:
```
lr = LogisticRegression(random_state=0)
```

In [ ]:
```
lr.fit(X_train,y_train)
```

In [ ]:
```
lr.coef_
```

In [ ]:
```
lr.intercept_
```

In [ ]:
```
y_pred = lr.predict(X_test)
```

In [ ]:
```
y_pred
```

In [ ]:
```
print(classification_report(y_test,y_pred))
```

In [ ]:
```
cm = confusion_matrix(y_test,y_pred)
cm
```

In [ ]:
```
plot_confusion_matrix(estimator=lr, X=X_test, y_true=y_test, cmap='YlGnBu')
plt.show()
```

In [ ]:
```
plot_roc_curve(estimator=lr, X=X_test, y=y_test)
plt.show()
```

In [ ]:
```

```

Python code done by Dennis Lam

In [ ]:
```

```