In [1]:

```
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns
from IPython.display import Image
from scipy import stats
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import OrdinalEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.tree import plot_tree
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.neighbors import NearestCentroid
from sklearn.neural_network import MLPClassifier
import time

import warnings
warnings.filterwarnings('ignore')
verbose = False
```

In [2]:

```
vehicles = pd.read_csv("data/dftRoadSafetyData_Vehicles_2018.csv")
vehicles.head()
```

Out[2]:

	Accident_Index	Vehicle_Reference	Vehicle_Type	Towing_and_Articulation	Vehicle_Manoeuvre
0	2018010080971	1	9	0	18
1	2018010080971	2	8	0	18
2	2018010080973	1	9	0	18
3	2018010080974	1	8	0	7
4	2018010080974	2	9	0	18

5 rows × 23 columns

In [3]:

```
accidents = pd.read_csv("data/dftRoadSafetyData_Accidents_2018.csv")
accidents.head()
```

Out[3]:

	Accident_Index	Location_Easting_OSGR	Location_Northing_OSGR	Longitude	Latitude	Poli
0	2018010080971	529150.0	182270.0	-0.139737	51.524587	
1	2018010080973	542020.0	184290.0	0.046471	51.539651	
2	2018010080974	531720.0	182910.0	-0.102474	51.529746	
3	2018010080981	541450.0	183220.0	0.037828	51.530179	
4	2018010080982	543580.0	176500.0	0.065781	51.469258	

5 rows × 32 columns

In [4]:

```python
casualties = pd.read_csv("data/dftRoadSafetyData_Casualties_2018.csv")
casualties.head()
```

Out[4]:

	Accident_Index	Vehicle_Reference	Casualty_Reference	Casualty_Class	Sex_of_Casualty	Age
0	2018010080971	1	1	2	2	
1	2018010080971	2	2	1	1	
2	2018010080973	1	1	3	1	
3	2018010080974	1	1	1	1	
4	2018010080981	1	1	1	1	

In [5]:

```python
df = vehicles.merge(accidents, how='inner', on='Accident_Index')
df = df.merge(casualties, how='inner', on='Accident_Index')
df.head()
```

Out[5]:

	Accident_Index	Vehicle_Reference_x	Vehicle_Type	Towing_and_Articulation	Vehicle_Manoeuvr
0	2018010135259	1	9	0	-
1	2018010135259	2	9	0	-
2	2018010135261	1	11	0	1
3	2018010135264	1	9	0	1
4	2018010135268	1	9	0	1

5 rows × 69 columns

In [6]:

```
population = pd.read_excel("data/regionalgrossdomesticproductgdplocalauthorities.xlsx", sh
eet_name=6, header=1, nrows=382)
population = population[["LA name", 2018]]
population = population.rename({2018:"Population"}, axis=1)
population.head()
```

Out[6]:

	LA name	Population
0	Hartlepool	93242
1	Middlesbrough	140545
2	Redcar and Cleveland	136718
3	Stockton-on-Tees	197213
4	Darlington	106566

In [7]:

```
gdp = pd.read_excel("data/regionalgrossdomesticproductgdplocalauthorities.xlsx", sheet_nam
e=7, header=1, nrows=382)
gdp = gdp[["LA name", '20183']]
gdp = gdp.rename({'20183':"GDP"}, axis=1)
gdp.head()
```

Out[7]:

	LA name	GDP
0	Hartlepool	18572
1	Middlesbrough	24103
2	Redcar and Cleveland	15793
3	Stockton-on-Tees	29843
4	Darlington	28866

In [8]:

```
gdp_growth = pd.read_excel("data/regionalgrossdomesticproductgdplocalauthorities.xlsx", sh
eet_name=13, header=1, nrows=382)
gdp_growth = gdp_growth[["LA name", '20183']]
gdp_growth = gdp_growth.rename({'20183':"GDP Growth"}, axis=1)
gdp_growth.head()
```

Out[8]:

	LA name	GDP Growth
0	Hartlepool	-2.6
1	Middlesbrough	3.5
2	Redcar and Cleveland	2.6
3	Stockton-on-Tees	-4.8
4	Darlington	-6.4

In [9]:

```
# Merge all financial Data
financial = pd.merge(population, gdp, on='LA name').merge(gdp_growth, on='LA name')
financial
```

Out[9]:

	LA name	Population	GDP	GDP Growth
0	Hartlepool	93242	18572	-2.6
1	Middlesbrough	140545	24103	3.5
2	Redcar and Cleveland	136718	15793	2.6
3	Stockton-on-Tees	197213	29843	-4.8
4	Darlington	106566	28866	-6.4
...	...	...	...	...
377	Lisburn and Castlereagh	144381	25918	3.6
378	Mid and East Antrim	138773	29885	-10.5
379	Mid Ulster	147392	24661	2.9
380	Newry, Mourne and Down	180012	18408	-3.0
381	Ards and North Down	160864	15034	0.1

382 rows × 4 columns

In [10]:

```
vlookup = pd.read_excel("data/variable lookup.xls", sheet_name=5, header=0)
vlookup = vlookup.rename({"label" :"LA name"},axis=1)
vlookup.head()
```

Out[10]:

	code	LA name
0	1	Westminster
1	2	Camden
2	3	Islington
3	4	Hackney
4	5	Tower Hamlets

In [11]:

```
# Add Names for LA
df = pd.merge(df, vlookup, left_on='Local_Authority_(District)', right_on='code')
df.head()
```

Out[11]:

	Accident_Index	Vehicle_Reference_x	Vehicle_Type	Towing_and_Articulation	Vehicle_Manoeuvr
0	2018010135259	1	9	0	-
1	2018010135259	2	9	0	-
2	2018010135309	1	9	0	1
3	2018010135309	2	11	0	1
4	2018010135320	1	1	-1	-

5 rows × 71 columns

In [12]:

```
# merge with financials
df = pd.merge(df, financial, on='LA name')
df.head()
```

Out[12]:

	Accident_Index	Vehicle_Reference_x	Vehicle_Type	Towing_and_Articulation	Vehicle_Manoeuvr
0	2018010135259	1	9	0	.
1	2018010135259	2	9	0	.
2	2018010135309	1	9	0	1
3	2018010135309	2	11	0	1
4	2018010135320	1	1	-1	.

5 rows × 74 columns

In [13]:

```
# Land Usage
land = pd.read_excel("data/Land_Use_England_2017.xlsx", sheet_name=3, header=8, nrows = 32
6)
land = land.iloc[:,[1,6,11]]
land = land.rename({"Local authority" : "LA name", "Unnamed: 6":"Residential", "Unnamed: 1
1":"Agriculture"}, axis = 1)
land.head()
```

Out[13]:

	LA name	Residential	Agriculture
0	Adur	4.226316	51.095201
1	Allerdale	0.282384	53.253852
2	Amber Valley	1.375413	67.644890
3	Arun	2.412087	55.505210
4	Ashfield	3.137516	44.549306

In [14]:

```
# merge with land usage
df = pd.merge(df, land, on='LA name')
df.head()
```

Out[14]:

	Accident_Index	Vehicle_Reference_x	Vehicle_Type	Towing_and_Articulation	Vehicle_Manoeuvr
0	2018010135259	1	9	0	-
1	2018010135259	2	9	0	-
2	2018010135309	1	9	0	1
3	2018010135309	2	11	0	1
4	2018010135320	1	1	-1	-

5 rows × 76 columns

In [15]:

```
# Final Set of Input Columns
df.columns
```

Out[15]:

```
Index(['Accident_Index', 'Vehicle_Reference_x', 'Vehicle_Type',
       'Towing_and_Articulation', 'Vehicle_Manoeuvre',
       'Vehicle_Location-Restricted_Lane', 'Junction_Location',
       'Skidding_and_Overturning', 'Hit_Object_in_Carriageway',
       'Vehicle_Leaving_Carriageway', 'Hit_Object_off_Carriageway',
       '1st_Point_of_Impact', 'Was_Vehicle_Left_Hand_Drive?',
       'Journey_Purpose_of_Driver', 'Sex_of_Driver', 'Age_of_Driver',
       'Age_Band_of_Driver', 'Engine_Capacity_(CC)', 'Propulsion_Code',
       'Age_of_Vehicle', 'Driver_IMD_Decile', 'Driver_Home_Area_Type',
       'Vehicle_IMD_Decile', 'Location_Easting_OSGR', 'Location_Northing_OSG
R',
       'Longitude', 'Latitude', 'Police_Force', 'Accident_Severity',
       'Number_of_Vehicles', 'Number_of_Casualties', 'Date', 'Day_of_Week',
       'Time', 'Local_Authority_(District)', 'Local_Authority_(Highway)',
       '1st_Road_Class', '1st_Road_Number', 'Road_Type', 'Speed_limit',
       'Junction_Detail', 'Junction_Control', '2nd_Road_Class',
       '2nd_Road_Number', 'Pedestrian_Crossing-Human_Control',
       'Pedestrian_Crossing-Physical_Facilities', 'Light_Conditions',
       'Weather_Conditions', 'Road_Surface_Conditions',
       'Special_Conditions_at_Site', 'Carriageway_Hazards',
       'Urban_or_Rural_Area', 'Did_Police_Officer_Attend_Scene_of_Accident',
       'LSOA_of_Accident_Location', 'Vehicle_Reference_y',
       'Casualty_Reference', 'Casualty_Class', 'Sex_of_Casualty',
       'Age_of_Casualty', 'Age_Band_of_Casualty', 'Casualty_Severity',
       'Pedestrian_Location', 'Pedestrian_Movement', 'Car_Passenger',
       'Bus_or_Coach_Passenger', 'Pedestrian_Road_Maintenance_Worker',
       'Casualty_Type', 'Casualty_Home_Area_Type', 'Casualty_IMD_Decile',
       'code', 'LA name', 'Population', 'GDP', 'GDP Growth', 'Residential',
       'Agriculture'],
      dtype='object')
```

In [16]:

```
# TODO - Fatalities (always) and older graphs
# TODO : Cleanup
# TODO  : Missing
# TODO  : PCA
# TODO : System Specs, GPU
# TODO : https://www3.cs.stonybrook.edu/~anshul/vis18_poster.pdf
```

Descriptive Statistics

In [17]:

```
df = df.replace({-1:np.nan})
df.describe()
```

Out[17]:

	Vehicle_Reference_x	Vehicle_Type	Towing_and_Articulation	Vehicle_Manoeuvre	Vehicle_ Restric
count	232974.00000	232672.000000	231133.000000	228821.000000	2288
mean	1.65297	10.333560	0.031506	12.938358	
std	2.25571	10.898588	0.314701	6.185495	
min	1.00000	1.000000	0.000000	1.000000	
25%	1.00000	9.000000	0.000000	7.000000	
50%	1.00000	9.000000	0.000000	18.000000	
75%	2.00000	9.000000	0.000000	18.000000	
max	999.00000	98.000000	5.000000	18.000000	

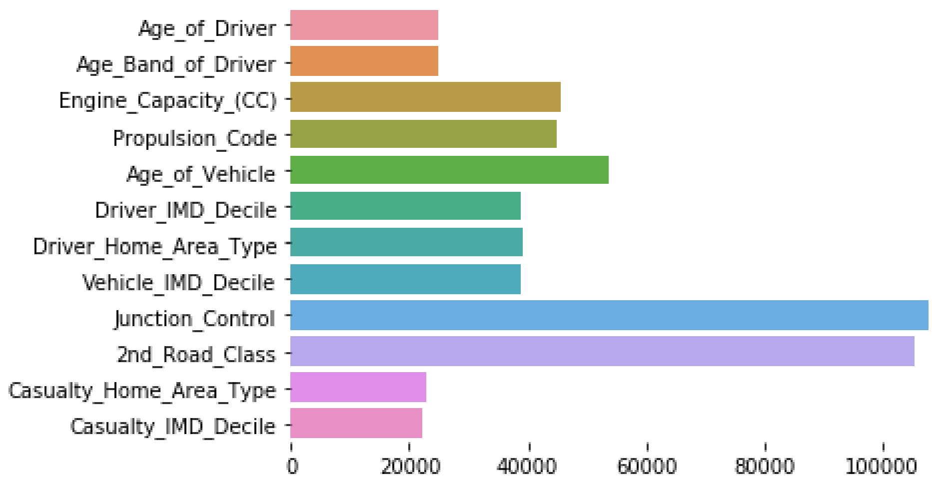
8 rows × 70 columns

In [18]:

```
# Add New Variables
df['Month'] = pd.to_datetime(df.Date).dt.month
df['Hour'] = pd.to_datetime(df.Time).dt.hour
df['Older_Driver'] = (df['Age_of_Driver']>38)
```

In [19]:

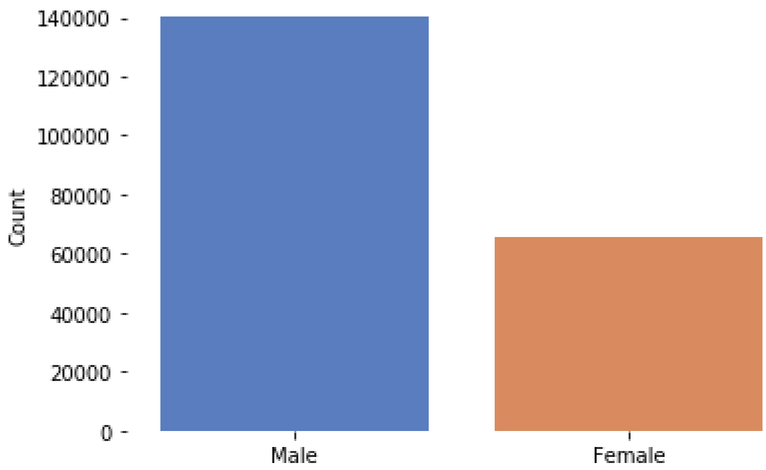
```
missing = df.isna().sum()
missing = missing[missing > 5000]
sns.set_palette("muted")
ax = sns.barplot(y=missing.index, x=missing)
sns.despine(left=True, bottom=True)

# Drop Driver_IMD_Decile, Vehicle_IMD_Decile
df = df.drop(['Driver_IMD_Decile', 'Vehicle_IMD_Decile'], axis=1)
# Remove Unknown Gender
df = df[df['Sex_of_Driver'] != 3]
# Remove drivers with no age
df = df[df['Age_of_Driver'].notnull()]
```



EDA

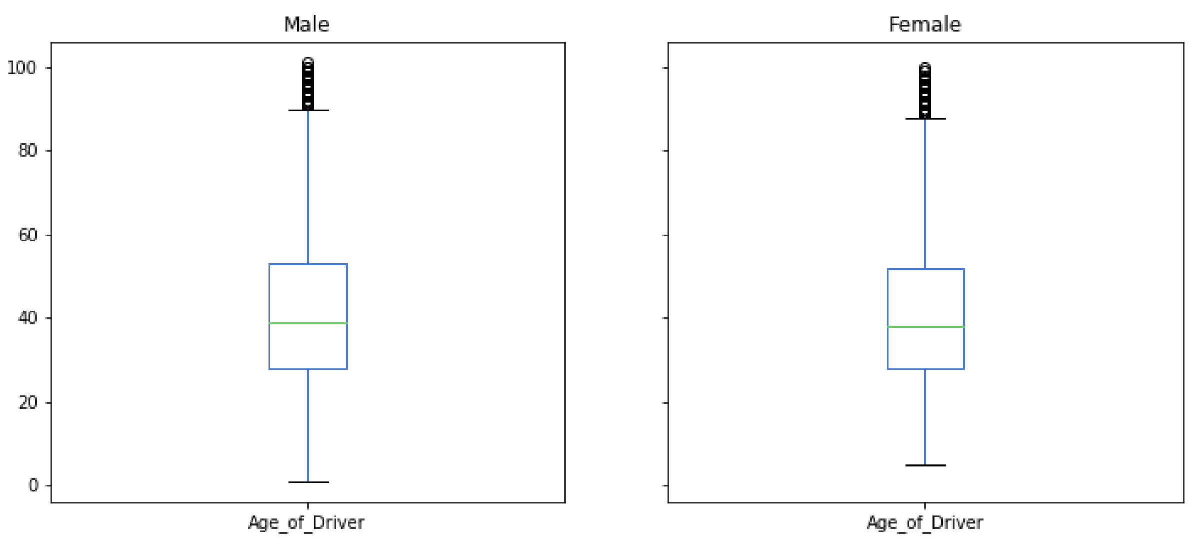
In [20]:

```
# EDA of Gender
gender = df['Sex_of_Driver'].value_counts().rename({1.0:"Male", 2.0:"Female", 3.0:"Not Kno
wn"})
ax = sns.barplot(x=gender.index, y=gender)
ax.set(xlabel='', ylabel='Count')
sns.despine(left=True, bottom=True)
# Conclusion : More than 2x more Males than Females, need to be careful
# to make training data balanced between genders
```



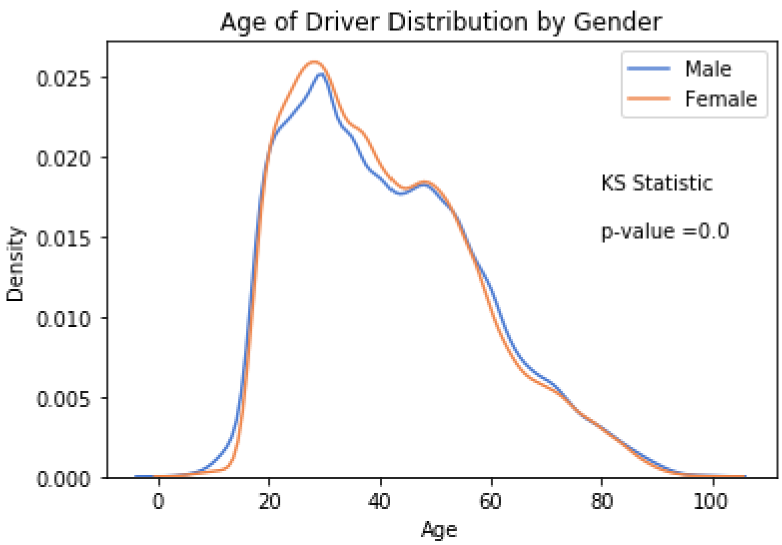
In [21]:

```
males = df[df['Sex_of_Driver']==1]
females = df[df['Sex_of_Driver']==2]
fig, ax = plt.subplots(ncols=2, figsize=(12, 5), sharey=True)
boxplot = males['Age_of_Driver'].plot(kind='box', ax=ax[0], title='Male')
boxplot = females['Age_of_Driver'].plot(kind='box', ax=ax[1], title='Female')
```



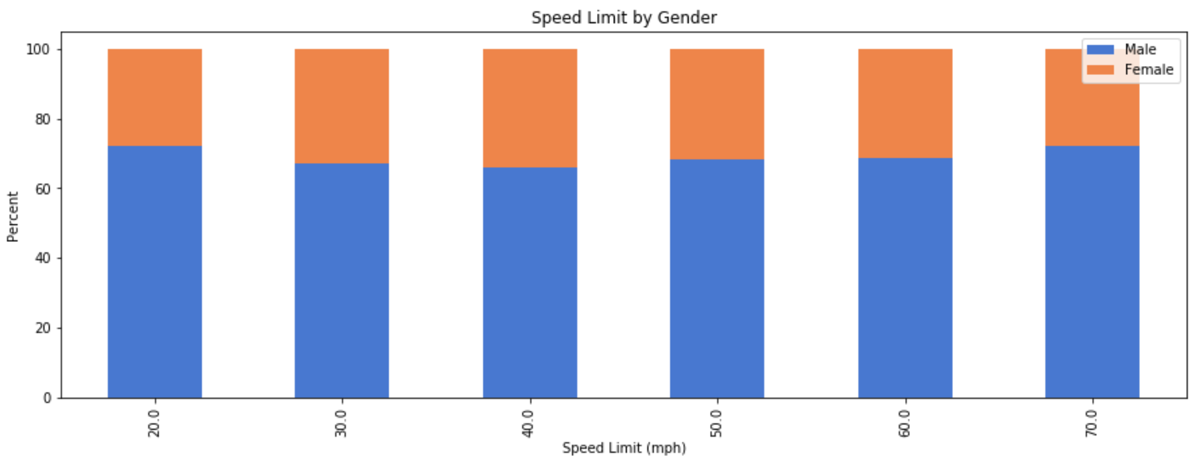
In [22]:

```
pvalue = stats.ks_2samp(males['Age_of_Driver'], females['Age_of_Driver']).pvalue
# Histogram of Age and Gender
fig, ax = plt.subplots()
sns.distplot(males['Age_of_Driver'], label='Male',hist=False, bins=range(1, 110, 10), ax=a
x, kde=True)
sns.distplot(females['Age_of_Driver'], label='Female',hist=False, bins=range(1, 110, 10),
ax=ax, kde=True)
ax.set(xlabel='Age', ylabel='Density')
ax.set_title("Age of Driver Distribution by Gender")
ax.text(80.0, 0.015, "KS Statistic\n\np-value =" + str(round(pvalue,2)))
plt.show()
```



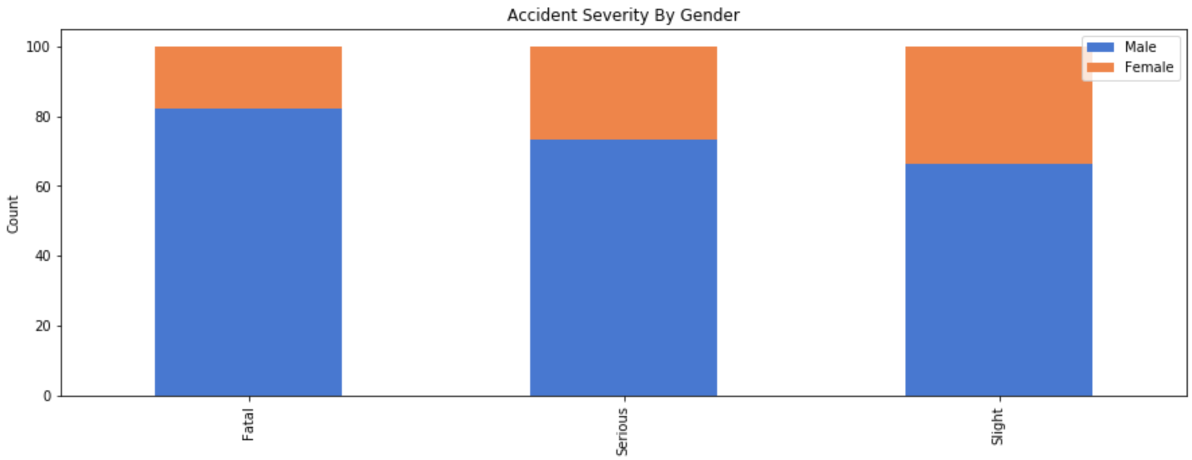
In [23]:

```
data = df.pivot_table(index='Speed_limit', columns='Sex_of_Driver', values='Road_Type', ag
gfunc='count')
data = data.dropna()
data = data.div(data.sum(1), axis=0) * 100
ax = data.plot(kind="bar", figsize=(15,5), stacked=True)
ax.set_xlabel("Speed Limit (mph)")
ax.set_ylabel("Percent")
ax.set_title("Speed Limit by Gender")
ax.legend(["Male", "Female"]);
```



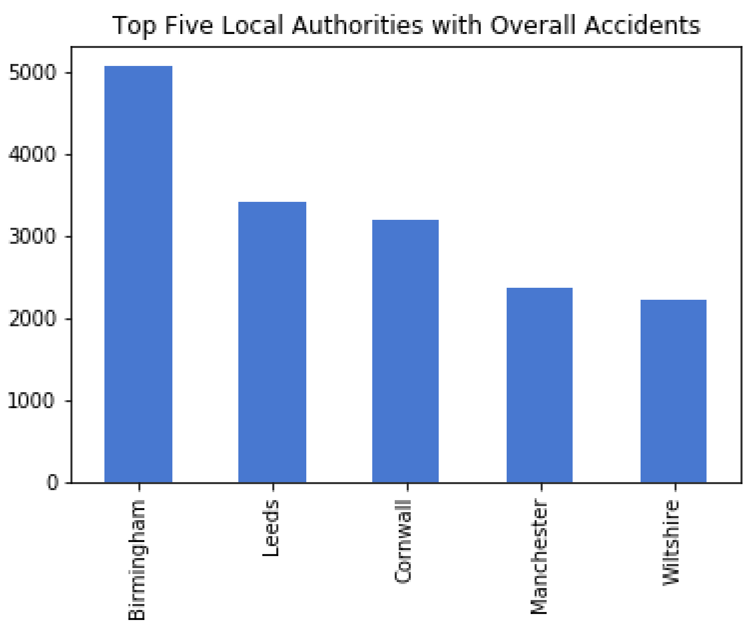
In [24]:

```
data = df.pivot_table(index='Accident_Severity', columns='Sex_of_Driver', values='Road_Typ
e', aggfunc='count')
data = data.dropna()
data = data.rename(index={1.0:"Fatal", 2.0: "Serious", 3.0: "Slight"})
data = data.div(data.sum(1), axis=0) * 100
ax = data.plot(kind="bar", figsize=(15,5), stacked=True)
ax.set_xlabel("")
ax.set_ylabel("Count")
ax.set_title("Accident Severity By Gender")
ax.legend(["Male", "Female"]);
```



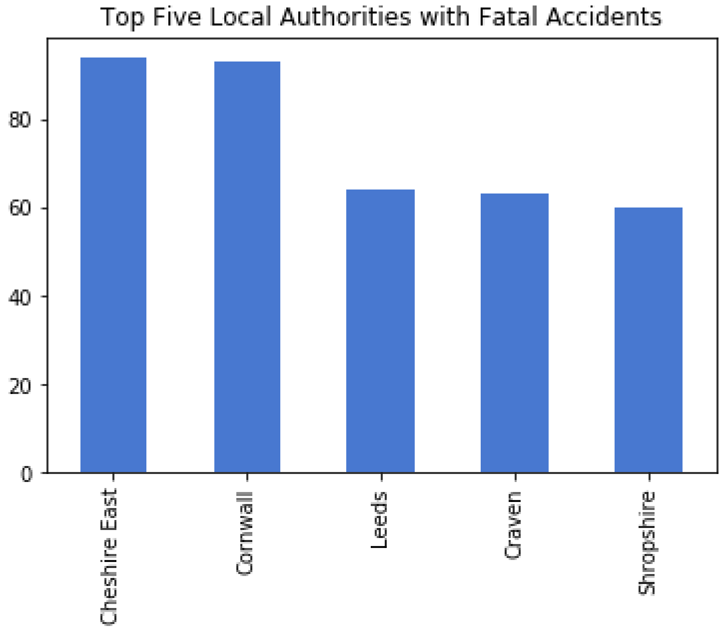
In [25]:

```
ax = df['LA name'].value_counts().iloc[0:5].plot(kind='bar')
x = ax.set_title("Top Five Local Authorities with Overall Accidents")
```



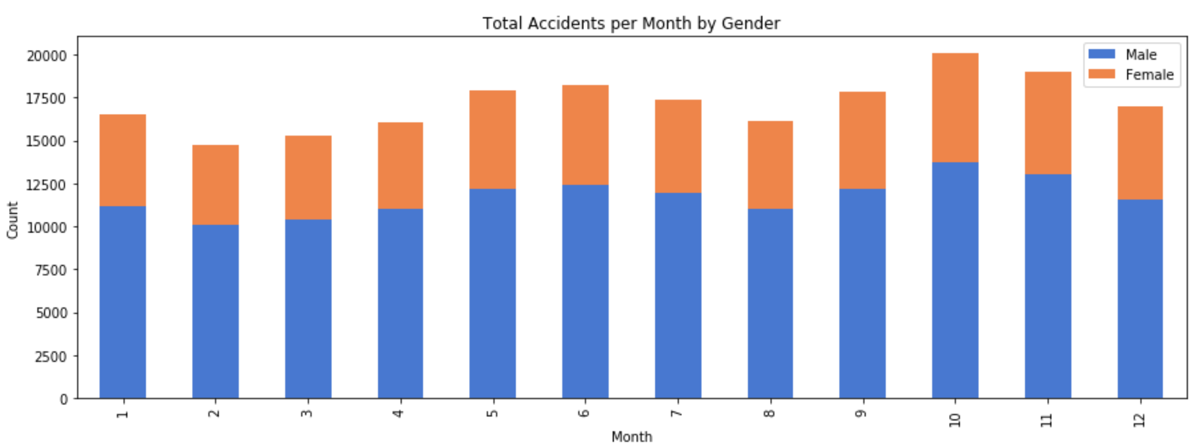
In [26]:

```
ax = df[df['Accident_Severity']==1]['LA name'].value_counts().iloc[0:5].plot(kind='bar')
x = ax.set_title("Top Five Local Authorities with Fatal Accidents")
```



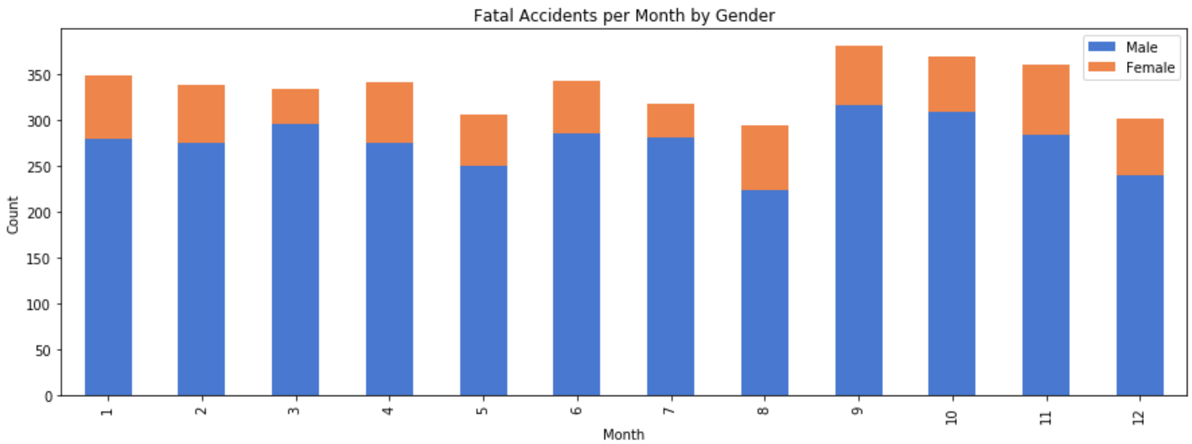
In [27]:

```
data = df.pivot_table(index='Month', columns='Sex_of_Driver', values='Road_Type', aggfunc=
'count')
data = data.dropna()
ax = data.plot(kind="bar", figsize=(15,5), stacked=True)
ax.set_xlabel("Month")
ax.set_ylabel("Count")
ax.set_title("Total Accidents per Month by Gender")
ax.legend(["Male", "Female"]);
```



In [28]:

```
data = df[df['Accident_Severity']==1].pivot_table(index='Month', columns='Sex_of_Driver',
values='Road_Type', aggfunc='count')
data = data.dropna()
ax = data.plot(kind="bar", figsize=(15,5), stacked=True)
ax.set_xlabel("Month")
ax.set_ylabel("Count")
ax.set_title("Fatal Accidents per Month by Gender")
ax.legend(["Male", "Female"]);
```

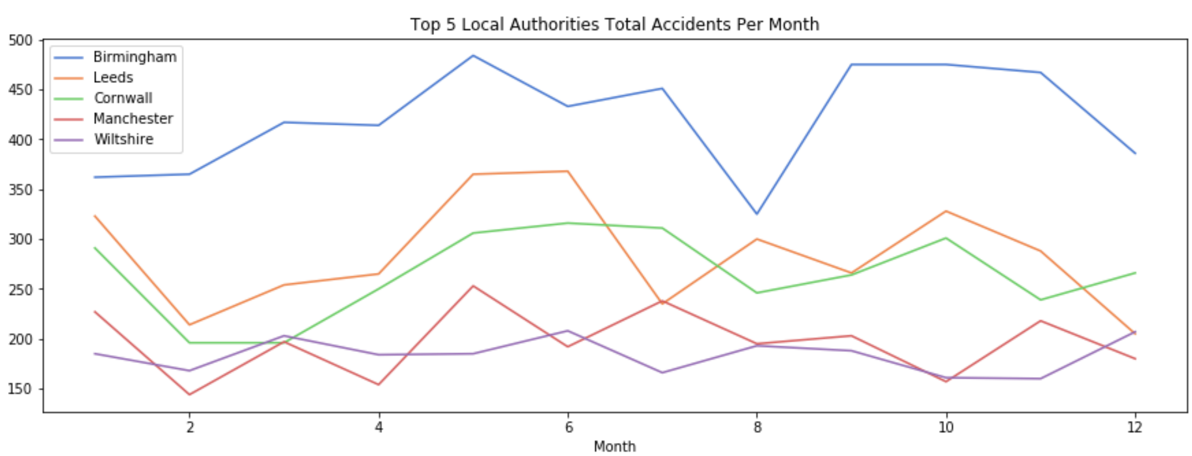


In [29]:

```
df[df["LA name"]== "Birmingham"].groupby("Month")["Road_Type"].count().plot(figsize=(15,5
))
df[df["LA name"]== "Leeds"].groupby("Month")["Road_Type"].count().plot()
df[df["LA name"]== "Cornwall"].groupby("Month")["Road_Type"].count().plot()
df[df["LA name"]== "Manchester"].groupby("Month")["Road_Type"].count().plot()
df[df["LA name"]== "Wiltshire"].groupby("Month")["Road_Type"].count().plot()
plt.legend(["Birmingham", "Leeds", "Cornwall", "Manchester", "Wiltshire"])
plt.title("Top 5 Local Authorities Total Accidents Per Month", loc='center', pad=None)
```

Out[29]:

```
Text(0.5, 1.0, 'Top 5 Local Authorities Total Accidents Per Month')
```

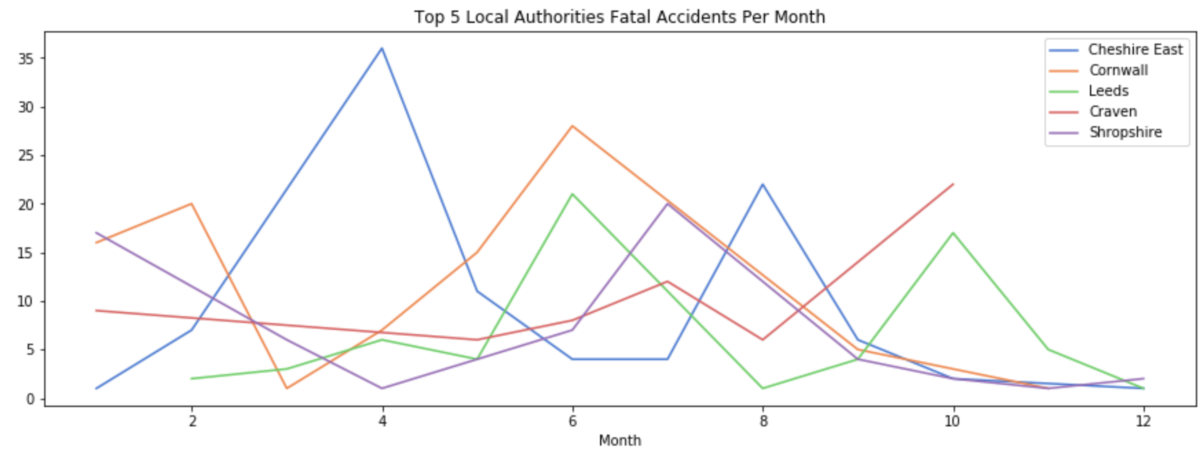


In [30]:

```
df2 = df[df['Accident_Severity']==1]
df2[df2["LA name"]== "Cheshire East"].groupby("Month")["Road_Type"].count().plot(figsize=(
15,5))
df2[df2["LA name"]== "Cornwall"].groupby("Month")["Road_Type"].count().plot()
df2[df2["LA name"]== "Leeds"].groupby("Month")["Road_Type"].count().plot()
df2[df2["LA name"]== "Craven"].groupby("Month")["Road_Type"].count().plot()
df2[df2["LA name"]== "Shropshire"].groupby("Month")["Road_Type"].count().plot()
plt.legend(["Cheshire East", "Cornwall", "Leeds", "Craven", "Shropshire"])
plt.title("Top 5 Local Authorities Fatal Accidents Per Month", loc='center', pad=None)
```

Out[30]:

```
Text(0.5, 1.0, 'Top 5 Local Authorities Fatal Accidents Per Month')
```

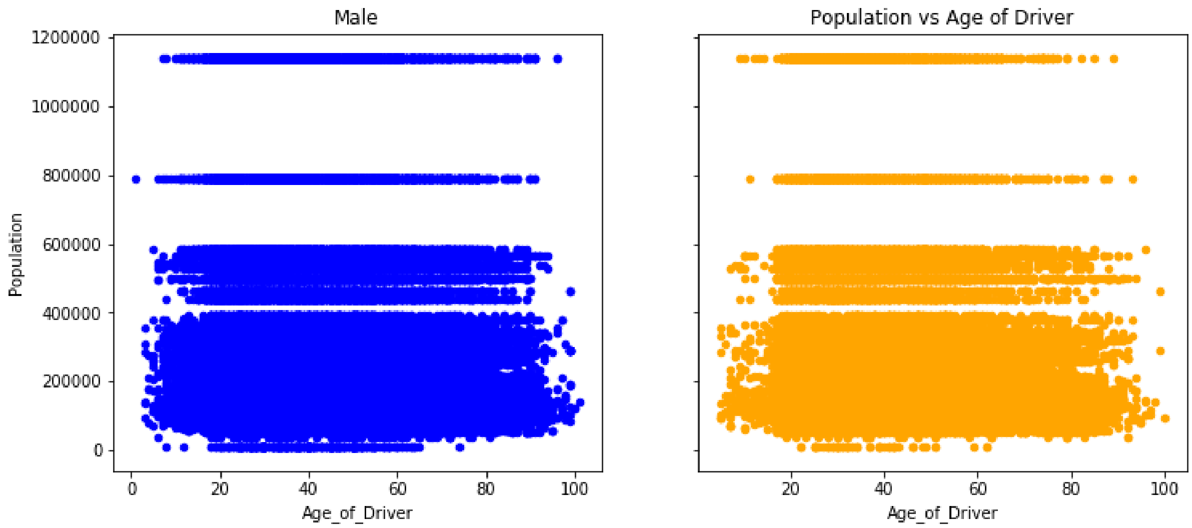


In [31]:

```
fig, ax = plt.subplots(ncols=2, figsize=(12, 5), sharey=True)
males.plot(kind='scatter', y='Population', x='Age_of_Driver', c='blue', ax=ax[0], title="M
ale")
females.plot(kind='scatter', y='Population', x='Age_of_Driver', c='orange', ax=ax[1], titl
e="Female")
plt.title("Population vs Age of Driver", loc='center', pad=None)
```

Out[31]:

```
Text(0.5, 1.0, 'Population vs Age of Driver')
```

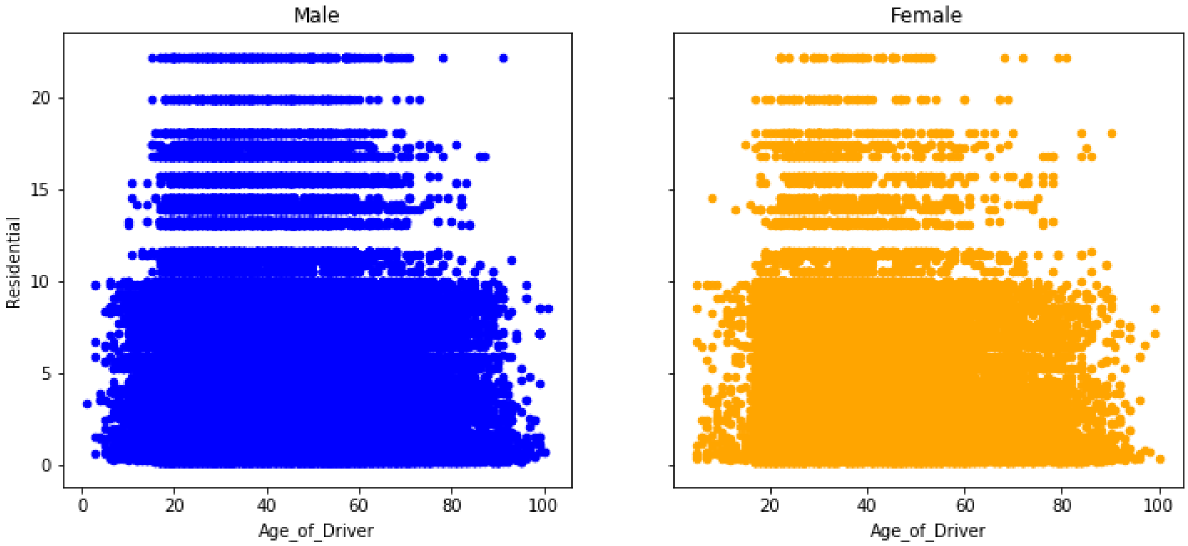


In [32]:

```
fig, ax = plt.subplots(ncols=2, figsize=(12, 5), sharey=True)
males.plot(kind='scatter', y='Residential', x='Age_of_Driver', c='blue', ax=ax[0], title=
"Male")
females.plot(kind='scatter', y='Residential', x='Age_of_Driver', c='orange', ax=ax[1], tit
le="Female")
```

Out[32]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18ed436af88>
```

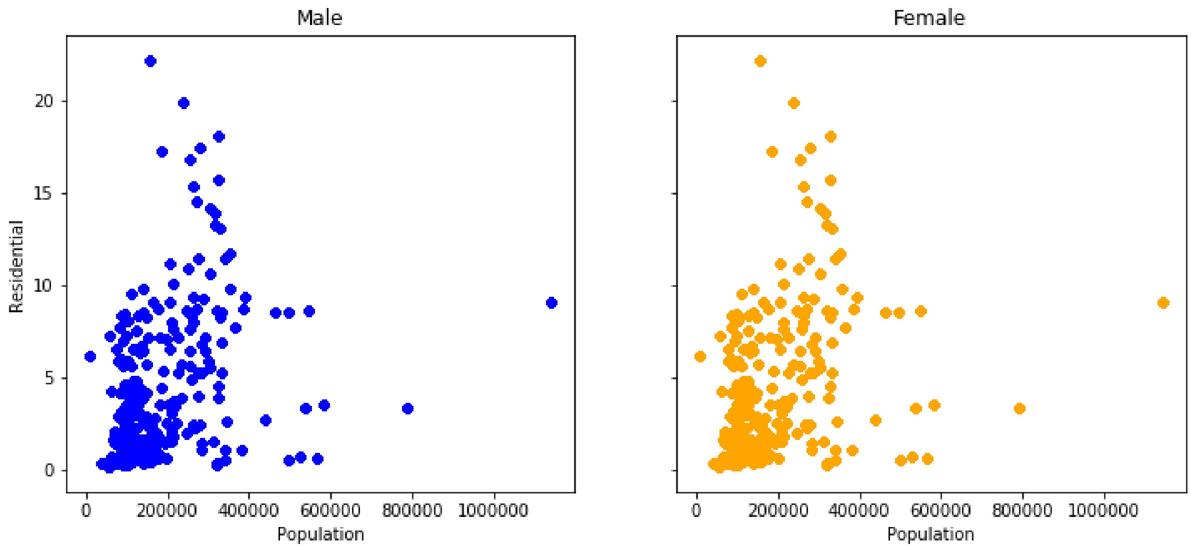


In [33]:

```
fig, ax = plt.subplots(ncols=2, figsize=(12, 5), sharey=True)
males.plot(kind='scatter', y='Residential', x='Population', c='blue', ax=ax[0], title="Mal
e")
females.plot(kind='scatter', y='Residential', x='Population', c='orange', ax=ax[1], title=
"Female")
```

Out[33]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18ed377f208>
```

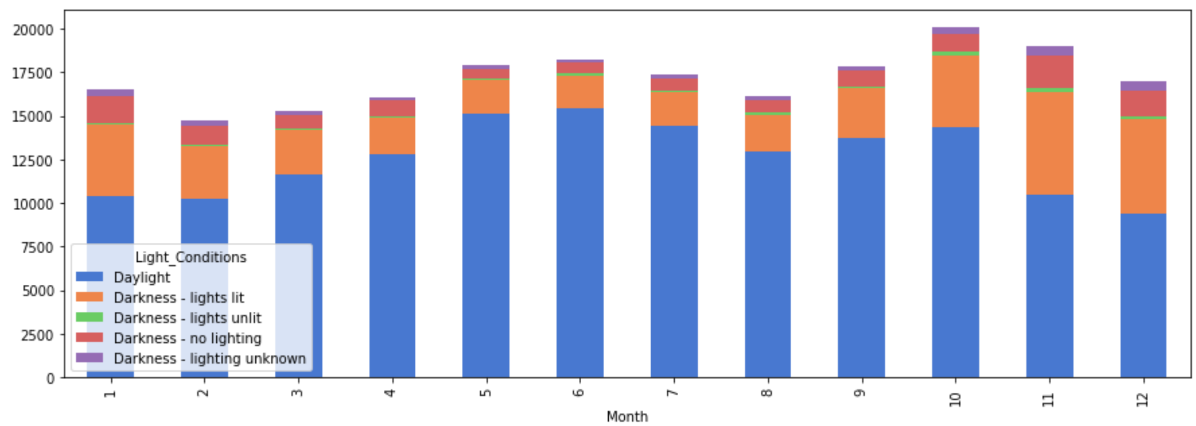


In [34]:

```
df2 = df.pivot_table(index="Month", columns="Light_Conditions", values="Road_Type", aggfun
c="count", fill_value=0)
df2 = df2.rename({1.0: "Daylight", 4.0: "Darkness - lights lit", 5.0: "Darkness - lights u
nlit", 6.0: "Darkness - no lighting", 7.0: "Darkness - lighting unknown"}, axis=1)
df2.plot(kind="bar", figsize=(15,5), stacked=True)
```

Out[34]:

<matplotlib.axes._subplots.AxesSubplot at 0x18ed37fd648>

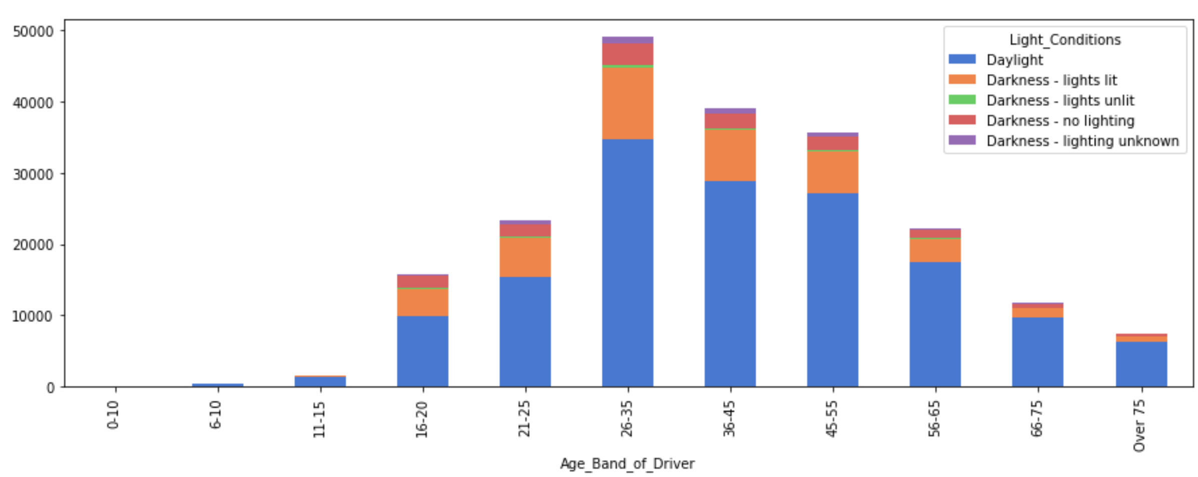


In [35]:

```
df2 = df.pivot_table(index="Age_Band_of_Driver", columns="Light_Conditions", values="Road_
Type", aggfunc="count", fill_value=0)
df2 = df2.rename({1.0: "0-10", 2.0: "6-10", 3.0: "11-15", 4.0: "16-20",
                  5.0: "21-25", 6.0: "26-35", 7.0: "36-45", 8.0:"45-55", 9.0:"56-65", 10.0
:"66-75", 11.0:"Over 75"})
df2 = df2.rename({1.0: "Daylight", 4.0: "Darkness - lights lit", 5.0: "Darkness - lights u
nlit", 6.0: "Darkness - no lighting", 7.0: "Darkness - lighting unknown"}, axis=1)
df2.plot(kind="bar", figsize=(15,5), stacked=True)
```

Out[35]:

<matplotlib.axes._subplots.AxesSubplot at 0x18ed8b0e4c8>



In [36]:

```
# TODO : Time and Daytime etc, Road_Surface_Conditions etc
df = df[df.Towing_and_Articulation.notnull()]
df = df[df.Vehicle_Manoeuvre.notnull()]
df = df[df.Skidding_and_Overturning.notnull()]
df = df[df.Journey_Purpose_of_Driver.notnull()]
df = df[df.Casualty_Type.notnull()]
df = df[df.Sex_of_Casualty.notnull()]
df = df[df.Road_Surface_Conditions.notnull()]
df = df[df['GDP Growth'].notnull()]
df = df[df.Age_of_Casualty.notnull()]
df = df[df.Vehicle_Type.notnull()]
df = df[df.Hour.notnull()]

print("Number of datapoints after removal of missing data = ", df.shape[0])
```

```
Number of datapoints after removal of missing data =  198414
```

Pipeline

In [37]:

```
# z-scale for quantative data
def z_scale(x):
    return (x-x.mean())/x.std()
def nop(x):
    return x

# TODO : Binary : Is Weekend?

# column transformer
column_transformer = ColumnTransformer(
    transformers = [
    ('Towing_and_Articulation', OneHotEncoder(categories='auto'), ["Towing_and_Articulatio
n"]),
    ('Vehicle_Type', OneHotEncoder(categories='auto'), ["Vehicle_Type"]),
    ('Vehicle_Manoeuvre', OneHotEncoder(categories='auto'), ["Vehicle_Manoeuvre"]),
    ('Skidding_and_Overturning', OneHotEncoder(categories='auto'), ["Skidding_and_Overturn
ing"]),
    ('Journey_Purpose_of_Driver', OneHotEncoder(categories='auto'), ["Journey_Purpose_of_D
river"]),
    ('LA name', OneHotEncoder(categories='auto'), ["LA name"]),
    ('Hour', OneHotEncoder(categories='auto'), ["Hour"]),
    ('Road_Type', OneHotEncoder(categories='auto'), ["Road_Type"]),
    ('Month', OrdinalEncoder(), ["Month"]),
    ('Sex_of_Casualty', OneHotEncoder(categories='auto'), ["Sex_of_Casualty"]),
    ('Sex_of_Driver', OneHotEncoder(categories='auto'), ["Sex_of_Driver"]),
    ('Number_of_Casualties', MinMaxScaler(), ["Number_of_Casualties"]),
    ('Weather_Conditions', OneHotEncoder(categories='auto'), ["Weather_Conditions"]),
    ('Road_Surface_Conditions', OneHotEncoder(categories='auto'), ["Road_Surface_Condition
s"]),
    ('Light_Conditions', OneHotEncoder(categories='auto'), ["Light_Conditions"]),
    ('Urban_or_Rural_Area', OneHotEncoder(categories='auto'), ["Urban_or_Rural_Area"]),
    ('Population', MinMaxScaler(), ["Population"]),
    ('GDP', FunctionTransformer(func=z_scale, validate=False), ["GDP"]),
    ('GDP Growth', FunctionTransformer(func=nop, validate=False), ["GDP Growth"]),
    ('Age_of_Casualty', FunctionTransformer(func=nop, validate=False), ["Age_of_Casualty"
]),
    ('Agriculture', MinMaxScaler(), ["Agriculture"]),
    ('Residential', MinMaxScaler(), ["Residential"]),
    #('Age_of_Driver', FunctionTransformer(func=nop, validate=False), ["Age_of_Driver"]),
    ],
    remainder = 'drop')

def show_confusion_result(title, x, y):
    start = time.time()
    y_pred = pipeline.predict(x)
    end = time.time()
    # create confusion matrix
    cf_matrix = confusion_matrix(y_true=y, y_pred=y_pred)
    TN, FP, FN, TP = cf_matrix.ravel()
    precision = TP/(TP+FP)
    recall = TP/(TP+FN)
    F1 = 2*(precision*recall)/(precision+recall)
    accuracy = (TP+TN)/(TN + FP + FN + TP)
    # Work Cited: https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
    group_names = ['True Neg','False Pos','False Neg','True Pos']
```

```
    group_counts = ["{0:0.0f}".format(value) for value in cf_matrix.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in cf_matrix.flatten()/np.sum(c
f_matrix)]
    labels = [f"{v1}\n\n{v2}\n\n{v3}" for v1, v2, v3 in zip(group_names,group_counts,group
_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    fig, ax = plt.subplots()
    sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues', ax=ax)
    ax.set_ylim([0,2])
    ax.text(2.7, 1.4, 'Accuracy  ' + str(round(accuracy*100,2)) + '%', c='blue')
    ax.text(2.7, 1.2, 'Precision  ' + str(round(precision*100,2)) + '%', c='blue')
    ax.text(2.7, 1.0, 'Recall       ' + str(round(recall*100,2)) + '%', c='blue' )
    ax.text(2.7, 0.8, 'F1           ' + str(round(F1*100,2)) + '%', c='blue' )
    x =plt.title(title)
    return [accuracy, precision, recall, F1, end-start]


def make_summary_table(result):
    results_summary = (pd.DataFrame(np.array(result), columns=['Accuracy %', 'Precision %'
, 'Recall %', 'F1 %', 'Prediction Time (s)'])*100.0).round(2)
    results_summary['Settings'] = ['Default', 'Default', 'Default', 'Optimized', 'Optimize
d', 'Optimized']
    results_summary['Data'] = ['Train', 'Validate', 'Test', 'Train', 'Validate', 'Test']
    results_summary = results_summary[['Settings', 'Data', 'Accuracy %', 'Precision %', 'R
ecall %', 'F1 %', 'Prediction Time (s)']]
    return results_summary

fit_time = []
```

Train (60), Validate (20), Test (20) Split

In [38]:

```
label = 'Older_Driver'
# split it into training, validation, and test splits, with a 60/20/20%
x_train_validate, x_test, y_train_validate, y_test = train_test_split(df.drop(label, axis=
1), df[label], test_size=0.2, random_state=42)
# now split train_validate into train (75%) and validate (25%)
x_train, x_validate, y_train, y_validate = train_test_split(x_train_validate, y_train_vali
date, test_size=0.25, random_state=42)
# set number of folds for validation:
nfolds = 5
```

Decision Tree Classifier (default)

In [39]:

```
decisiontree = DecisionTreeClassifier()
pipeline = Pipeline([
    ('preprocessing', column_transformer),
    ('decisiontree', decisiontree)
], verbose=verbose)
```

In [40]:

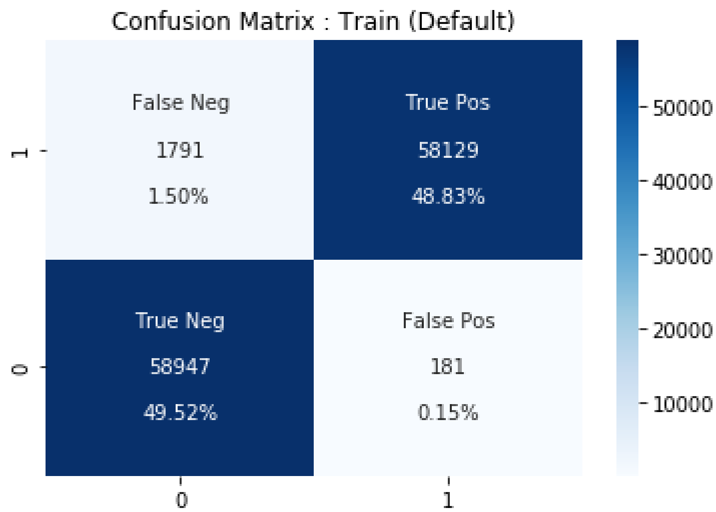
```
# Train
pipeline.fit(x_train, y_train)
# Accuracy
train_score = accuracy_score(y_train, pipeline.predict(x_train), normalize=True, sample_we
ight=None)
test_score = accuracy_score(y_test, pipeline.predict(x_test), normalize=True, sample_weigh
t=None)
train_score, test_score
```

Out[40]:

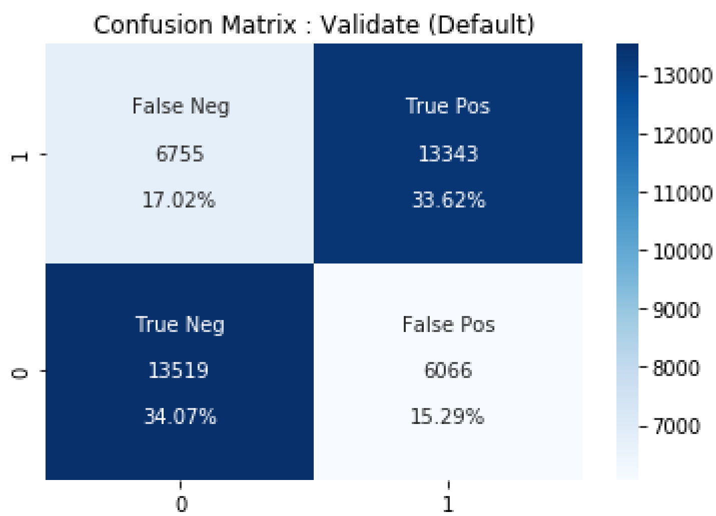
```
(0.9834352530071904, 0.6800141118363027)
```

In [41]:

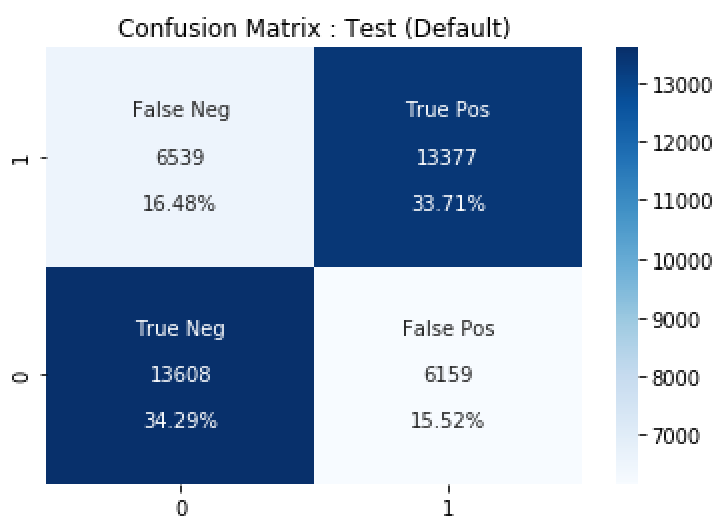
```
result = []
result.append(show_confusion_result('Confusion Matrix : Train (Default)', x_train, y_train
))
result.append(show_confusion_result('Confusion Matrix : Validate (Default)', x_validate, y
_validate))
result.append(show_confusion_result('Confusion Matrix : Test (Default)', x_test, y_test))
```



Accuracy 98.34%
Precision 99.69%
Recall 97.01%
F1 98.33%



Accuracy 67.69%
Precision 68.75%
Recall 66.39%
F1 67.55%



Accuracy 68.0%
Precision 68.47%
Recall 67.17%
F1 67.81%

Decision Tree Classifier Hyper-Parameter Tuning

In [42]:

```
criterion = ['gini', 'entropy']
splitter = ['best', 'random']
max_depth = [6,10,20]
parameters = dict(decisiontree__criterion=criterion,
                  decisiontree__max_depth=max_depth,
                  decisiontree__splitter = splitter)
clf = GridSearchCV(pipeline, parameters, cv=nfolds)
# Fit the grid search
clf.fit(x_validate, y_validate)
print('Best criterion:', clf.best_estimator_.get_params()['decisiontree__criterion'])
print('Best max_depth:', clf.best_estimator_.get_params()['decisiontree__max_depth'])
print('Best splitter:', clf.best_estimator_.get_params()['decisiontree__splitter'])
```

```
Best criterion: gini
Best max_depth: 6
Best splitter: best
```

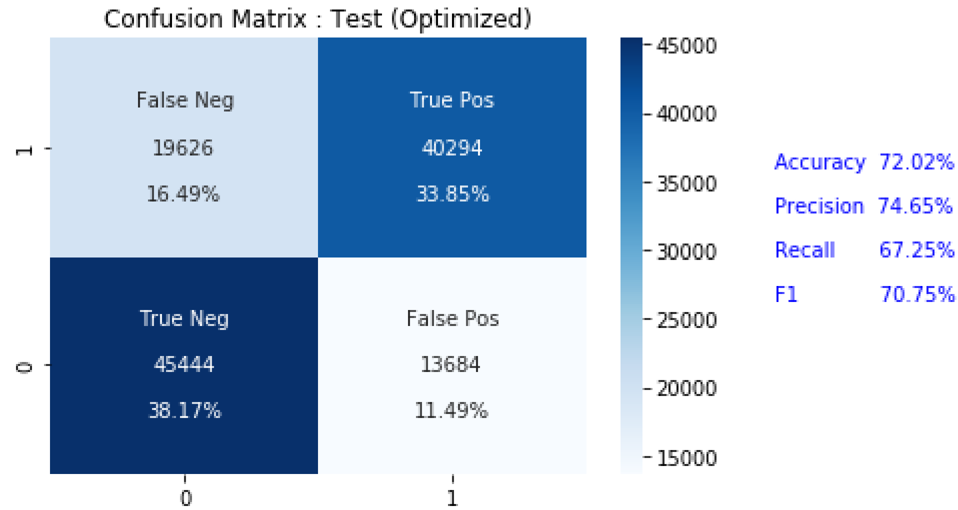
In [43]:

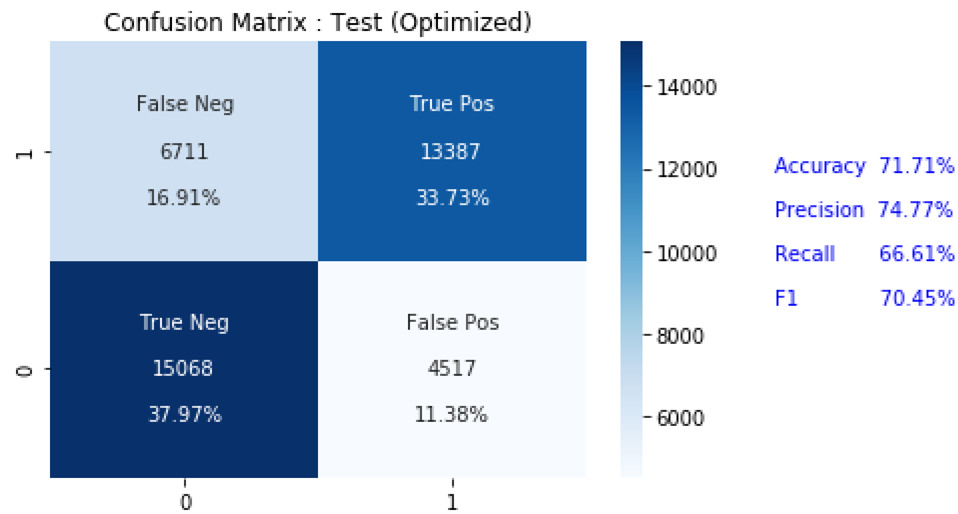
```
decisiontree = clf.best_estimator_.get_params()['decisiontree']
pipeline = Pipeline([
    ('preprocessing', column_transformer),
    ('decisiontree', decisiontree)
], verbose=verbose)
# Train
start = time.time();
pipeline.fit(x_train, y_train);
fit_time.append(time.time()-start);

# Accuracy
train_score = accuracy_score(y_train, pipeline.predict(x_train), normalize=True, sample_we
ight=None)
print("Training Data Accuracy =", train_score, " Test Data Accuracy =", test_score)
```

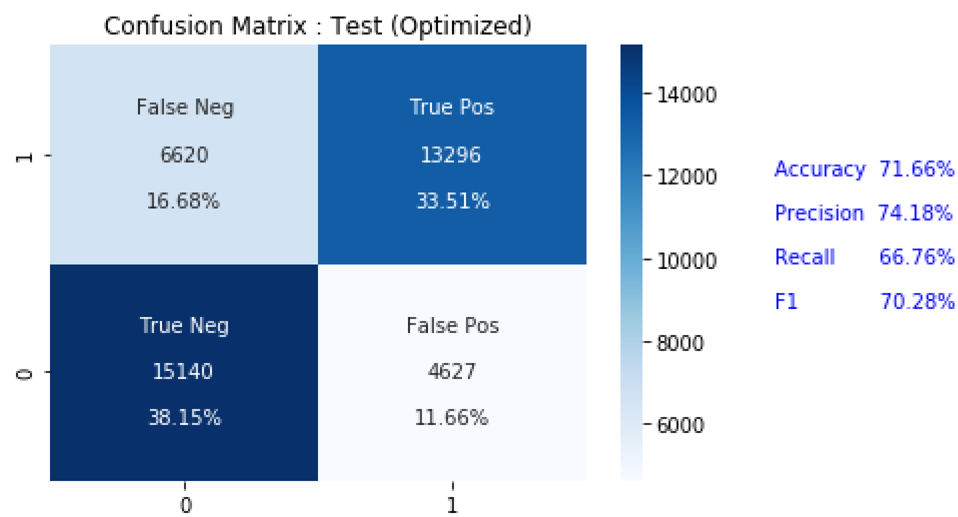
```
Training Data Accuracy = 0.7201968953699348  Test Data Accuracy = 0.680014111
8363027
```

In [44]:

```
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_train, y_trai
n))
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_validate, y_v
alidate))
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_test, y_test
))
```


Confusion Matrix : Test (Optimized)
False Neg
19626
16.49%
True Pos
40294
33.85%
True Neg
45444
38.17%
False Pos
13684
11.49%
0
1
0
1
45000
40000
35000
30000
25000
20000
15000
Accuracy 72.02%
Precision 74.65%
Recall 67.25%
F1 70.75%


Confusion Matrix : Test (Optimized)
False Neg
6711
16.91%
True Pos
13387
33.73%
True Neg
15068
37.97%
False Pos
4517
11.38%
0
1
0
1
14000
12000
10000
8000
6000
Accuracy 71.71%
Precision 74.77%
Recall 66.61%
F1 70.45%


Confusion Matrix : Test (Optimized)
False Neg
6620
16.68%
True Pos
13296
33.51%
True Neg
15140
38.15%
False Pos
4627
11.66%
0
1
0
1
14000
12000
10000
8000
6000
Accuracy 71.66%
Precision 74.18%
Recall 66.76%
F1 70.28%

Decision Tree Final Results

In [45]:

```
decisiontree_df = make_summary_table(result)
decisiontree_df
```

Out[45]:

	Settings	Data	Accuracy %	Precision %	Recall %	F1 %	Prediction Time (s)
0	Default	Train	98.34	99.69	97.01	98.33	37.19
1	Default	Validate	67.69	68.75	66.39	67.55	15.62
2	Default	Test	68.00	68.47	67.17	67.81	17.89
3	Optimized	Train	72.02	74.65	67.25	70.75	37.84
4	Optimized	Validate	71.71	74.77	66.61	70.45	14.69
5	Optimized	Test	71.66	74.18	66.76	70.28	15.62

In [ ]:

```

```

Random Forest Classifier (default)

In [46]:

```
randomforest = RandomForestClassifier(n_estimators=5,max_depth=5)
pipeline = Pipeline([
    ('preprocessing', column_transformer),
    ('randomforest', randomforest)
], verbose=verbose)
```

In [47]:

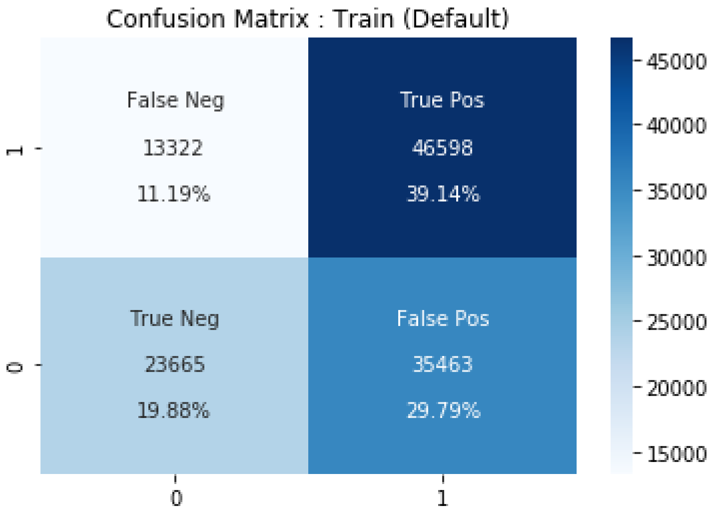
```
# Train
pipeline.fit(x_train, y_train)
# Accuracy
train_score = accuracy_score(y_train, pipeline.predict(x_train), normalize=True, sample_we
ight=None)
test_score = accuracy_score(y_test, pipeline.predict(x_test), normalize=True, sample_weigh
t=None)
train_score, test_score
```

Out[47]:

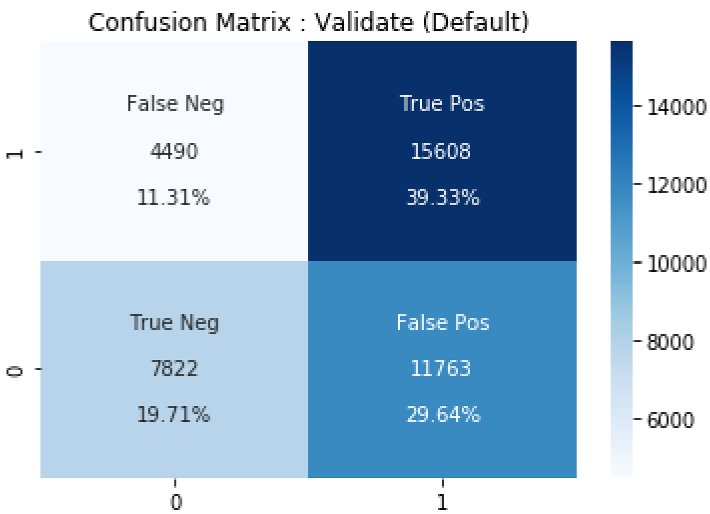
```
(0.5902073113366038, 0.5864475972078724)
```

In [48]:

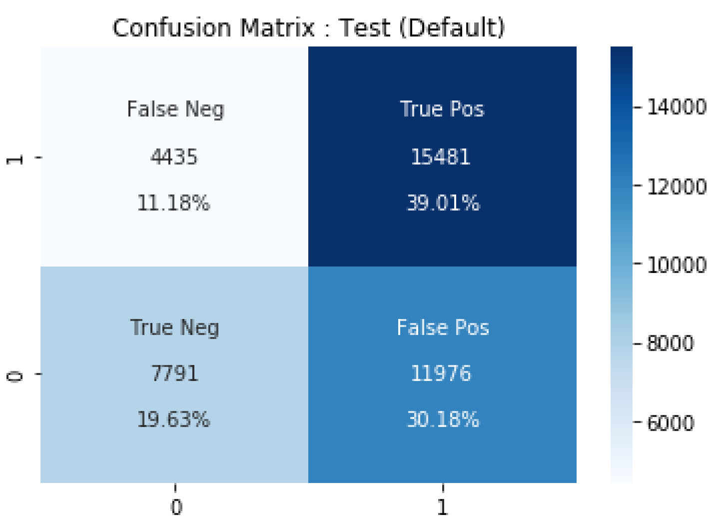
```
result = []
result.append(show_confusion_result('Confusion Matrix : Train (Default)', x_train, y_train
))
result.append(show_confusion_result('Confusion Matrix : Validate (Default)', x_validate, y
_validate))
result.append(show_confusion_result('Confusion Matrix : Test (Default)', x_test, y_test))
```



Accuracy 59.02%
Precision 56.78%
Recall 77.77%
F1 65.64%



Accuracy 59.04%
Precision 57.02%
Recall 77.66%
F1 65.76%



Accuracy 58.64%
Precision 56.38%
Recall 77.73%
F1 65.36%

Random Forest Classifier Hyper-Parameter Tuning

In [49]:

```
n_estimators = [5,20]
max_depth = [5,30]
criterion = ['entropy','gini']

parameters = dict(randomforest__n_estimators=n_estimators,
                  randomforest__max_depth=max_depth,
                 randomforest__criterion = criterion)
clf = GridSearchCV(pipeline, parameters, cv=nfolds)
# Fit the grid search
clf.fit(x_validate, y_validate)
print('Best n_estimators:', clf.best_estimator_.get_params()['randomforest__n_estimators'
])
print('Best max_depth:', clf.best_estimator_.get_params()['randomforest__max_depth'])
print('Best criterion:', clf.best_estimator_.get_params()['randomforest__criterion'])
```

```
Best n_estimators: 20
Best max_depth: 30
Best criterion: gini
```

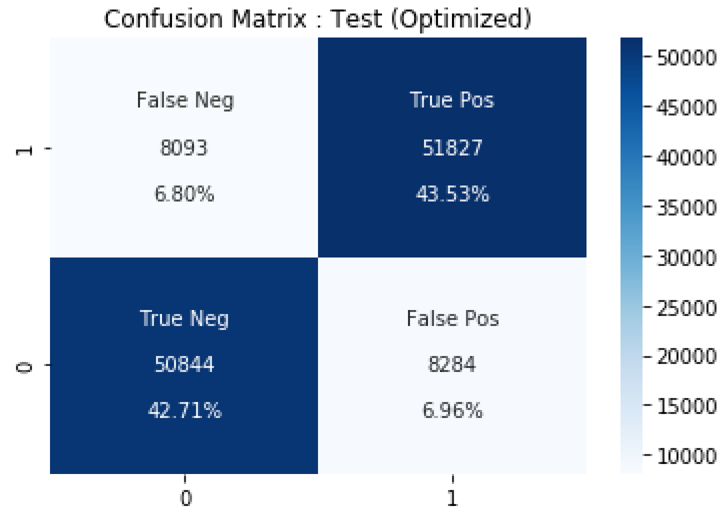
In [50]:

```
randomforest = clf.best_estimator_.get_params()['randomforest']
pipeline = Pipeline([
    ('preprocessing', column_transformer),
    ('randomforest', randomforest)
], verbose=verbose)
# Train
start = time.time();
pipeline.fit(x_train, y_train);
fit_time.append(time.time()-start);

# Accuracy
test_score = accuracy_score(y_test, pipeline.predict(x_test), normalize=True, sample_weigh
t=None)
test_score = accuracy_score(y_test, pipeline.predict(x_test), normalize=True, sample_weigh
t=None)
print("Training Data Accuracy =", train_score, " Test Data Accuracy =", test_score)
```

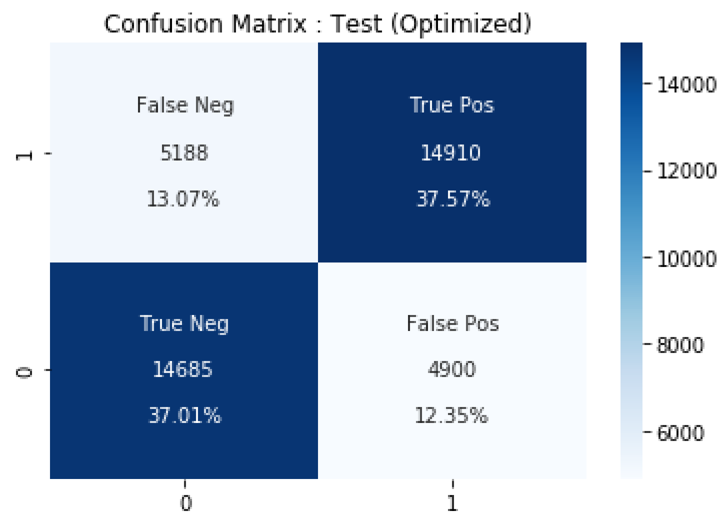
```
Training Data Accuracy = 0.5902073113366038  Test Data Accuracy = 0.745986946
55142
```

In [51]:

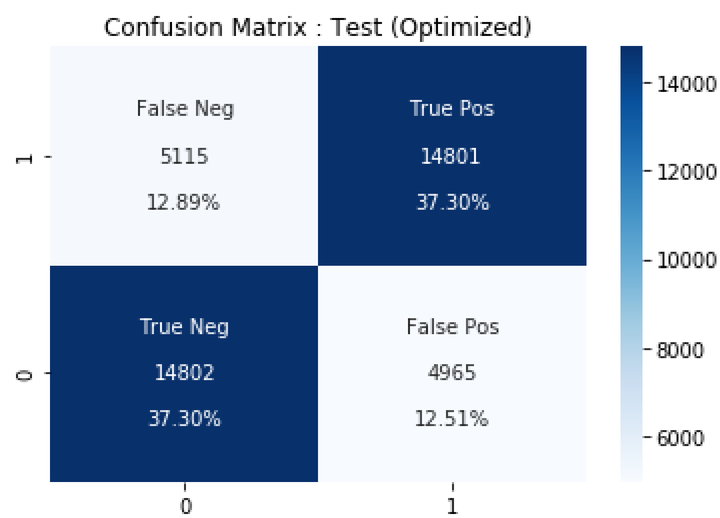
```
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_train, y_trai
n))
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_validate, y_v
alidate))
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_test, y_test
))
```



Accuracy 86.24%
Precision 86.22%
Recall 86.49%
F1 86.36%



Accuracy 74.58%
Precision 75.27%
Recall 74.19%
F1 74.72%



Accuracy 74.6%
Precision 74.88%
Recall 74.32%
F1 74.6%

Random Forest Final Results

In [52]:

```
randomforest_df = make_summary_table(result)
randomforest_df
```

Out[52]:

	Settings	Data	Accuracy %	Precision %	Recall %	F1 %	Prediction Time (s)
0	Default	Train	59.02	56.78	77.77	65.64	40.74
1	Default	Validate	59.04	57.02	77.66	65.76	15.56
2	Default	Test	58.64	56.38	77.73	65.36	17.19
3	Optimized	Train	86.24	86.22	86.49	86.36	97.95
4	Optimized	Validate	74.58	75.27	74.19	74.72	38.36
5	Optimized	Test	74.60	74.88	74.32	74.60	37.50

In [ ]:

```

```

SVC Classifier (default)

In [53]:

```
svc = SVC(gamma='auto', max_iter=10)
pipeline = Pipeline([
    ('preprocessing', column_transformer),
    ('svc', svc)
],
verbose=verbose)
```

In [54]:

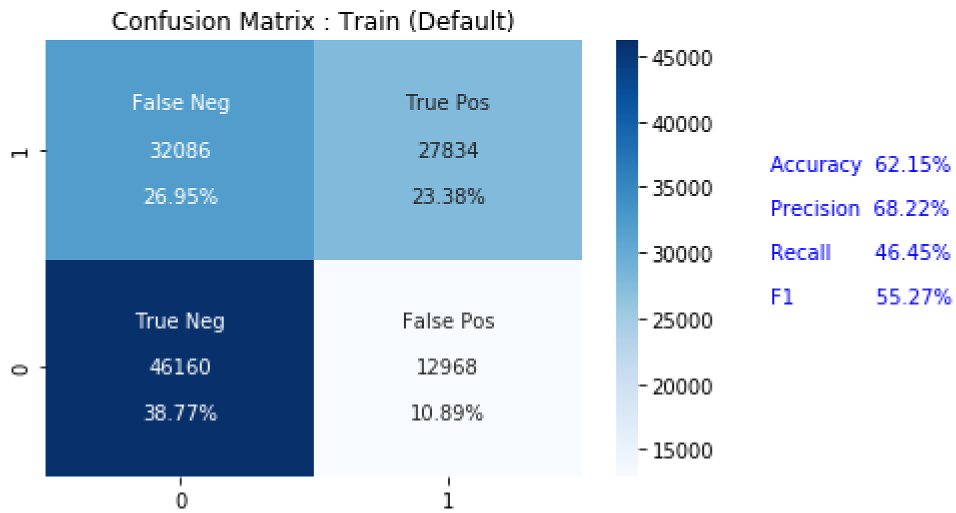
```
# Train
pipeline.fit(x_train, y_train)
# Accuracy
train_score = accuracy_score(y_train, pipeline.predict(x_train), normalize=True, sample_we
ight=None)
test_score = accuracy_score(y_test, pipeline.predict(x_test), normalize=True, sample_weigh
t=None)
train_score, test_score
```

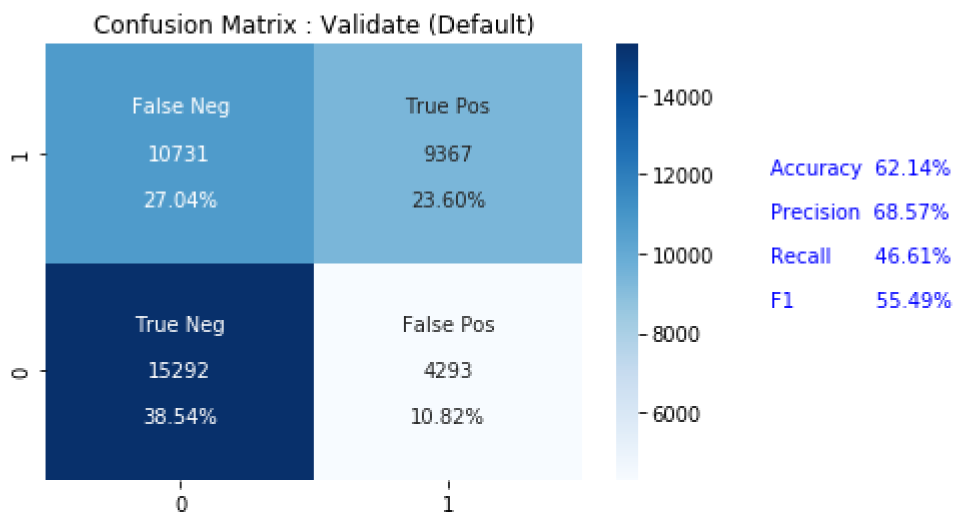
Out[54]:

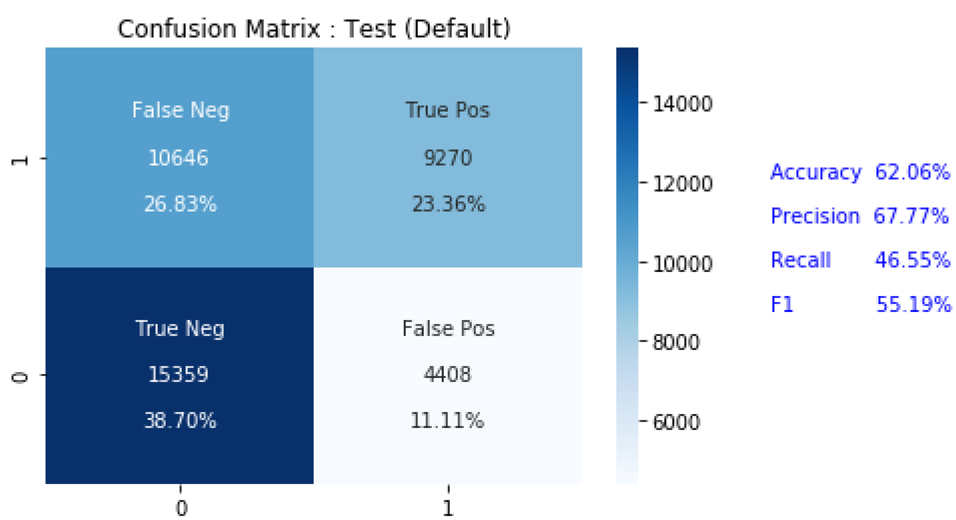
```
(0.6215476110476447, 0.6206436005342338)
```

In [55]:

```
result = []
result.append(show_confusion_result('Confusion Matrix : Train (Default)', x_train, y_train
))
result.append(show_confusion_result('Confusion Matrix : Validate (Default)', x_validate, y
_validate))
result.append(show_confusion_result('Confusion Matrix : Test (Default)', x_test, y_test))
```







SVC Classifier Results Hyper-Parameter Tuning

In [56]:

```
C = [(10 ** i) for i in range(-1,5)]

parameters = dict(svc__C=C)
clf = GridSearchCV(pipeline, parameters, cv=nfolds)
# Fit the grid search
clf.fit(x_validate, y_validate)
print('Best C:', clf.best_estimator_.get_params()['svc__C'])
```

```
Best C: 0.1
```

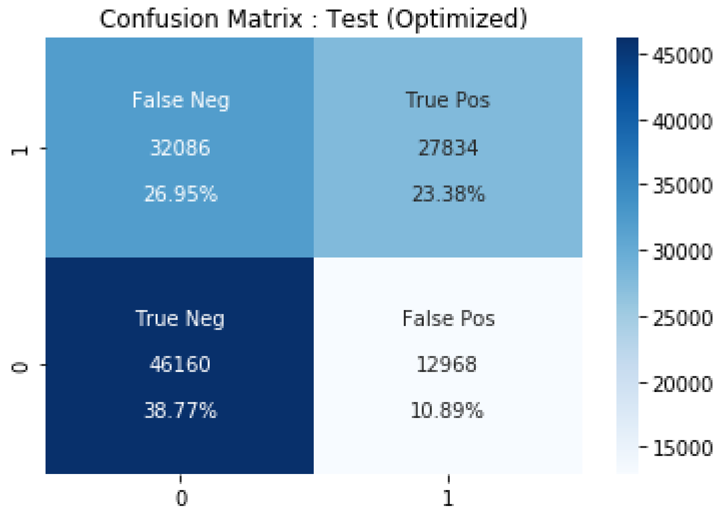
In [57]:

```
svc = clf.best_estimator_.get_params()['svc']
pipeline = Pipeline([
    ('preprocessing', column_transformer),
    ('svc', svc)
])
# Train
start = time.time();
pipeline.fit(x_train, y_train);
fit_time.append(time.time()-start);

# Accuracy
train_score = accuracy_score(y_train, pipeline.predict(x_train), normalize=True, sample_we
ight=None)
test_score = accuracy_score(y_test, pipeline.predict(x_test), normalize=True, sample_weigh
t=None)
print("Training Data Accuracy =", train_score, " Test Data Accuracy =", test_score)
```

```
Training Data Accuracy = 0.6215476110476447  Test Data Accuracy = 0.620643600
5342338
```

In [58]:

```
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_train, y_trai
n))
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_validate, y_v
alidate))
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_test, y_test
))
```

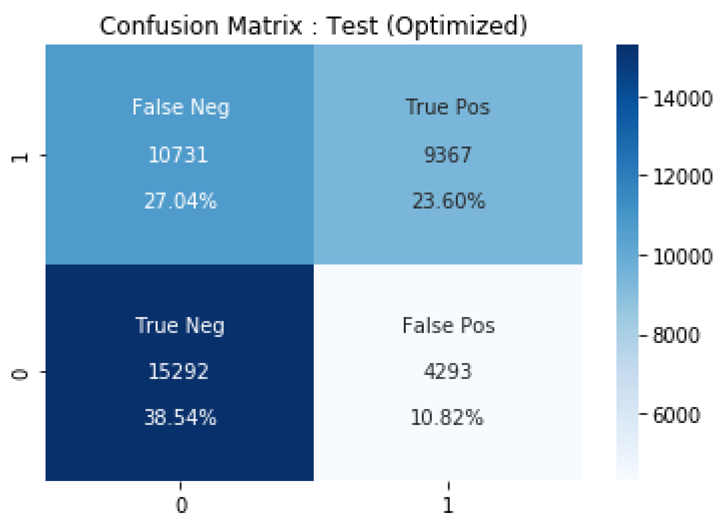


Accuracy 62.15%

Precision 68.22%

Recall 46.45%

F1 55.27%

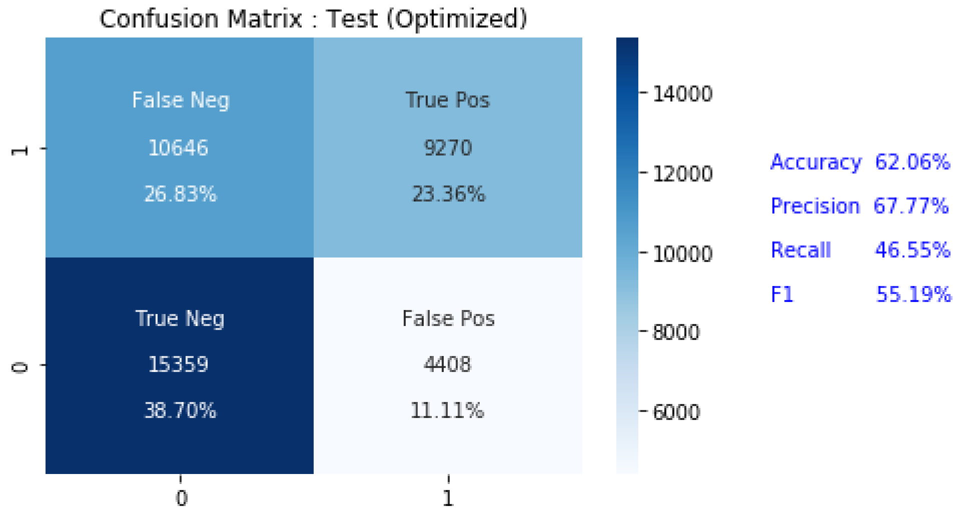


Accuracy 62.14%

Precision 68.57%

Recall 46.61%

F1 55.49%



SVC Final Results

In [59]:

```
svc_df = make_summary_table(result)
svc_df
```

Out[59]:

	Settings	Data	Accuracy %	Precision %	Recall %	F1 %	Prediction Time (s)
0	Default	Train	62.15	68.22	46.45	55.27	64.16
1	Default	Validate	62.14	68.57	46.61	55.49	23.66
2	Default	Test	62.06	67.77	46.55	55.19	24.91
3	Optimized	Train	62.15	68.22	46.45	55.27	54.91
4	Optimized	Validate	62.14	68.57	46.61	55.49	22.26
5	Optimized	Test	62.06	67.77	46.55	55.19	25.12

In [ ]:

```

```

Gradient Boost Classifier (default)

In [60]:

```
gboost = GradientBoostingClassifier()
pipeline = Pipeline([
    ('preprocessing', column_transformer),
    ('gboost', gboost)
], verbose=verbose)
```

In [61]:

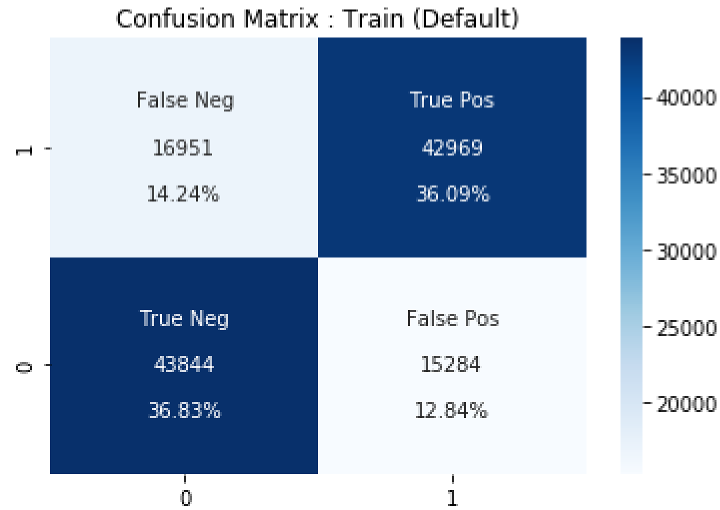
```
# Train
pipeline.fit(x_train, y_train)
# Accuracy
train_score = accuracy_score(y_train, pipeline.predict(x_train), normalize=True, sample_we
ight=None)
test_score = accuracy_score(y_test, pipeline.predict(x_test), normalize=True, sample_weigh
t=None)
train_score, test_score
```

Out[61]:

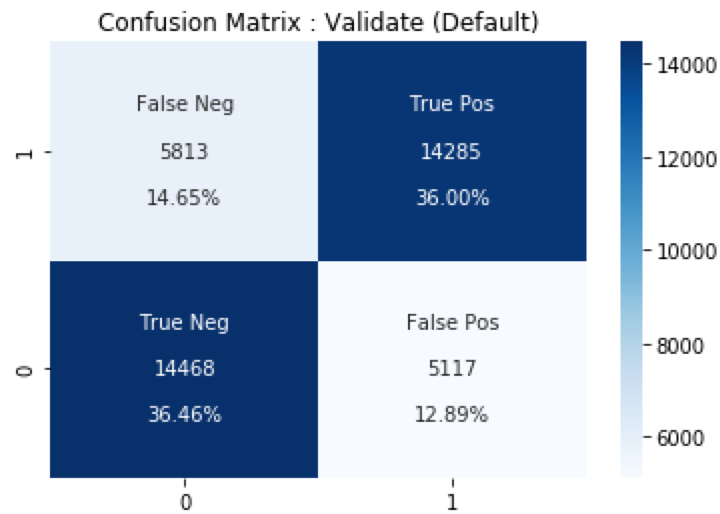
```
(0.7292268664740272, 0.725096388881889)
```

In [62]:

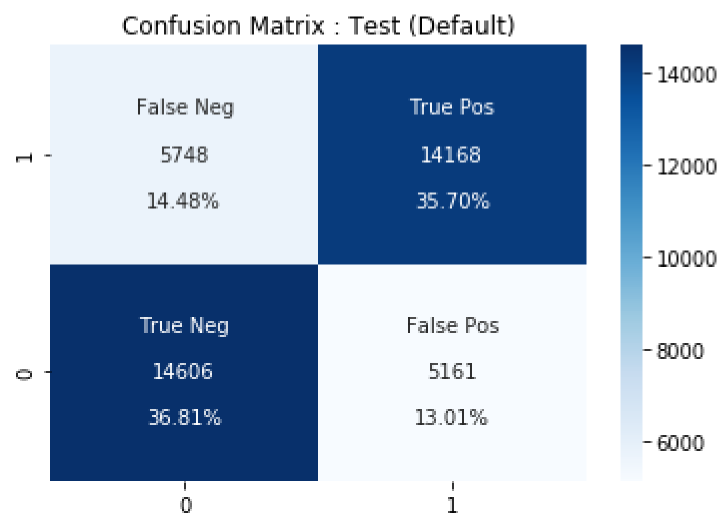
```
result = []
result.append(show_confusion_result('Confusion Matrix : Train (Default)', x_train, y_train
))
result.append(show_confusion_result('Confusion Matrix : Validate (Default)', x_validate, y
_validate))
result.append(show_confusion_result('Confusion Matrix : Test (Default)', x_test, y_test))
```



Accuracy 72.92%
Precision 73.76%
Recall 71.71%
F1 72.72%



Accuracy 72.46%
Precision 73.63%
Recall 71.08%
F1 72.33%



Accuracy 72.51%
Precision 73.3%
Recall 71.14%
F1 72.2%

Gradient Boost Hyper-Parameter Tuning

In [63]:

```
loss = ['deviance', 'exponential']
n_estimators = [50,100,200]

parameters = dict(gboost__n_estimators=n_estimators,
                  gboost__loss = loss)
clf = GridSearchCV(pipeline, parameters, cv=nfolds)
# Fit the grid search
clf.fit(x_validate, y_validate)
print('Best n_estimators:', clf.best_estimator_.get_params()['gboost__n_estimators'])
print('Best loss:', clf.best_estimator_.get_params()['gboost__loss'])
```

```
Best n_estimators: 200
Best loss: deviance
```

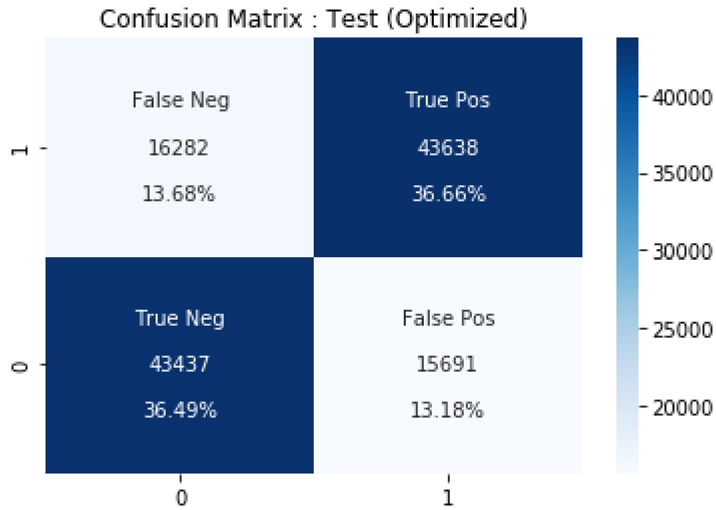
In [64]:

```
gboost = clf.best_estimator_.get_params()['gboost']
pipeline = Pipeline([
    ('preprocessing', column_transformer),
    ('gboost', gboost)
], verbose=verbose)
# Train
start = time.time();
pipeline.fit(x_train, y_train);
fit_time.append(time.time()-start);

# Accuracy
train_score = accuracy_score(y_train, pipeline.predict(x_train), normalize=True, sample_we
ight=None)
test_score = accuracy_score(y_test, pipeline.predict(x_test), normalize=True, sample_weigh
t=None)
print("Training Data Accuracy =", train_score, " Test Data Accuracy =", test_score)
```

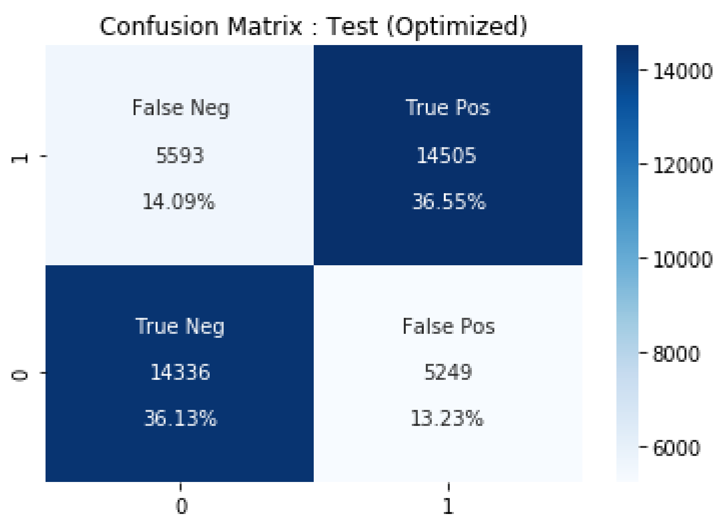
```
Training Data Accuracy = 0.7314276594314898  Test Data Accuracy = 0.726910767
8350931
```

In [65]:

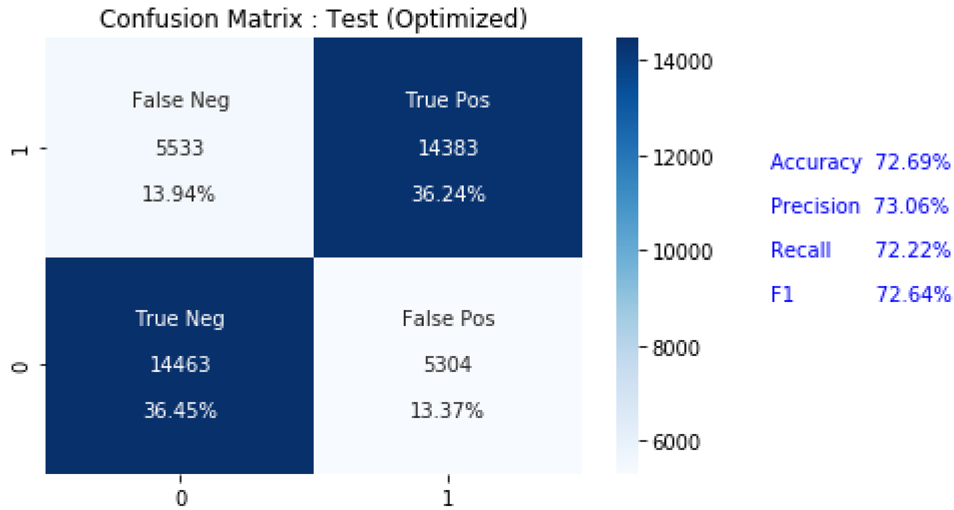
```
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_train, y_trai
n))
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_validate, y_v
alidate))
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_test, y_test
))
```



Accuracy 73.14%
Precision 73.55%
Recall 72.83%
F1 73.19%



Accuracy 72.68%
Precision 73.43%
Recall 72.17%
F1 72.79%



Gradient Boost Final Results

In [66]:

```
gboost_df = make_summary_table(result)
gboost_df
```

Out[66]:

	Settings	Data	Accuracy %	Precision %	Recall %	F1 %	Prediction Time (s)
0	Default	Train	72.92	73.76	71.71	72.72	53.27
1	Default	Validate	72.46	73.63	71.08	72.33	21.87
2	Default	Test	72.51	73.30	71.14	72.20	18.24
3	Optimized	Train	73.14	73.55	72.83	73.19	91.03
4	Optimized	Validate	72.68	73.43	72.17	72.79	31.50
5	Optimized	Test	72.69	73.06	72.22	72.64	26.56

In [ ]:

```

```

Bagging Classifier (default)

In [67]:

```
bagging = BaggingClassifier()
pipeline = Pipeline([
    ('preprocessing', column_transformer),
    ('bagging', bagging)
], verbose=verbose)
```

In [68]:

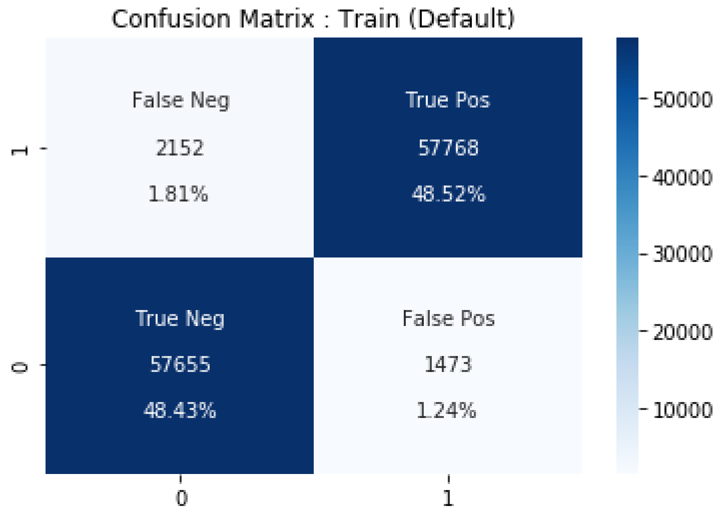
```
# Train
pipeline.fit(x_train, y_train)
# Accuracy
train_score = accuracy_score(y_train, pipeline.predict(x_train), normalize=True, sample_we
ight=None)
test_score = accuracy_score(y_test, pipeline.predict(x_test), normalize=True, sample_weigh
t=None)
train_score, test_score
```

Out[68]:

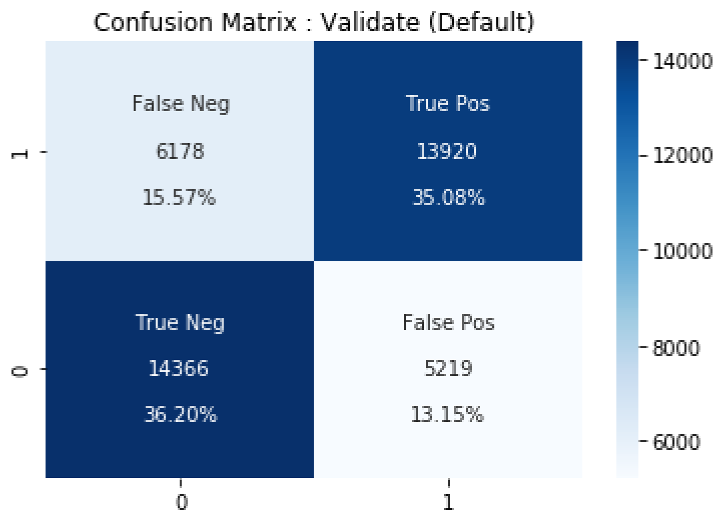
```
(0.9695500974396882, 0.7131265277322784)
```

In [69]:

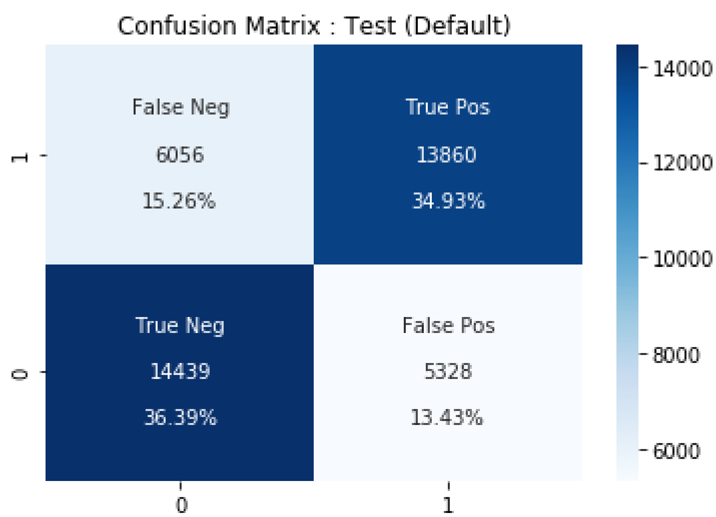
```
result = []
result.append(show_confusion_result('Confusion Matrix : Train (Default)', x_train, y_train
))
result.append(show_confusion_result('Confusion Matrix : Validate (Default)', x_validate, y
_validate))
result.append(show_confusion_result('Confusion Matrix : Test (Default)', x_test, y_test))
```



Accuracy 96.96%
Precision 97.51%
Recall 96.41%
F1 96.96%



Accuracy 71.28%
Precision 72.73%
Recall 69.26%
F1 70.95%



Accuracy 71.31%
Precision 72.23%
Recall 69.59%
F1 70.89%

Bagging Classifier Hyper-Parameter Tuning

In [70]:

```
max_samples = [0.5, 1.0, 1.5]
max_features = [0.5, 1.0, 1.5]
parameters = dict(bagging__max_features = max_features,
                  bagging__max_samples = max_samples)
clf = GridSearchCV(pipeline, parameters, cv=nfolds)
# Fit the grid search
clf.fit(x_validate, y_validate)
print('Best max_features:', clf.best_estimator_.get_params()['bagging__max_features'])
print('Best _max_samples:', clf.best_estimator_.get_params()['bagging__max_samples'])
```

```
Best max_features: 1.0
Best _max_samples: 0.5
```

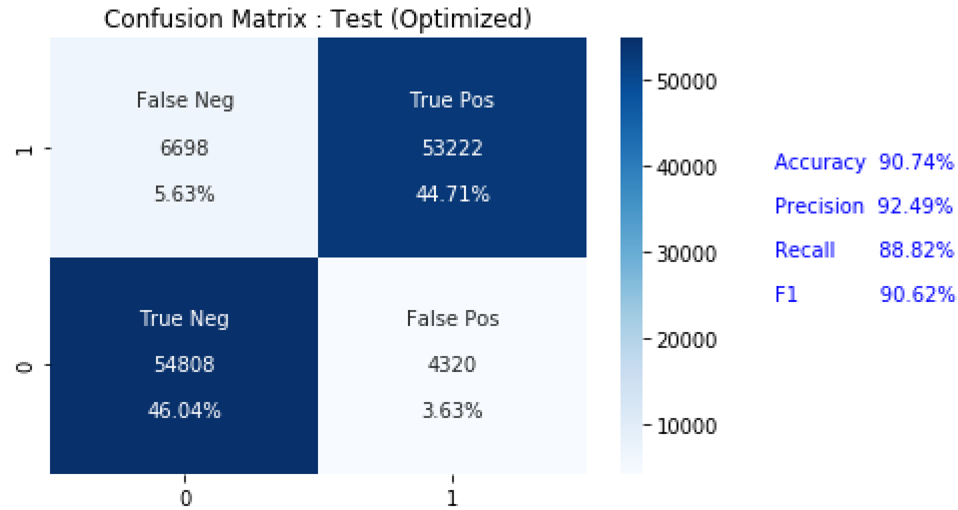
In [71]:

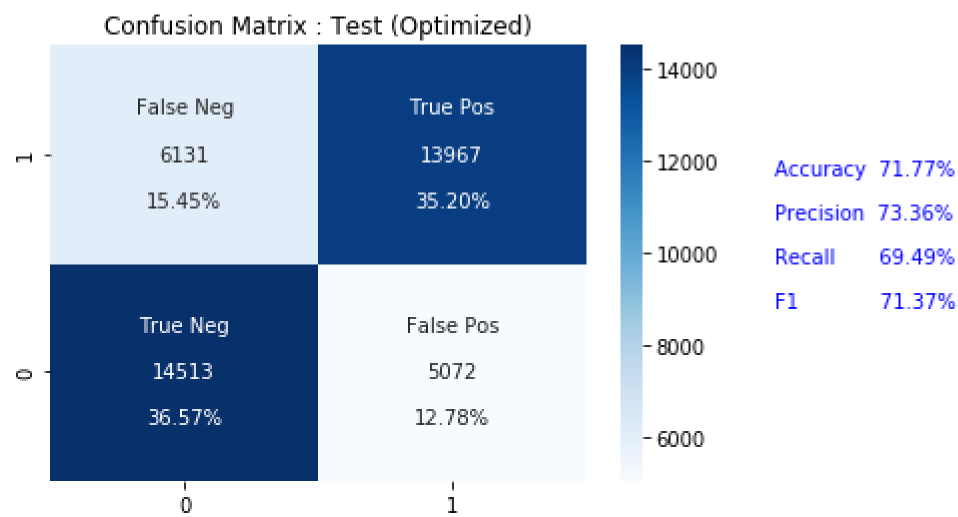
```
bagging = clf.best_estimator_.get_params()['bagging']
pipeline = Pipeline([
    ('preprocessing', column_transformer),
    ('bagging', bagging)
], verbose=verbose)
# Train
start = time.time();
pipeline.fit(x_train, y_train);
fit_time.append(time.time()-start);

# Accuracy
train_score = accuracy_score(y_train, pipeline.predict(x_train), normalize=True, sample_we
ight=None)
test_score = accuracy_score(y_test, pipeline.predict(x_test), normalize=True, sample_weigh
t=None)
print("Training Data Accuracy =", train_score, " Test Data Accuracy =", test_score)
```

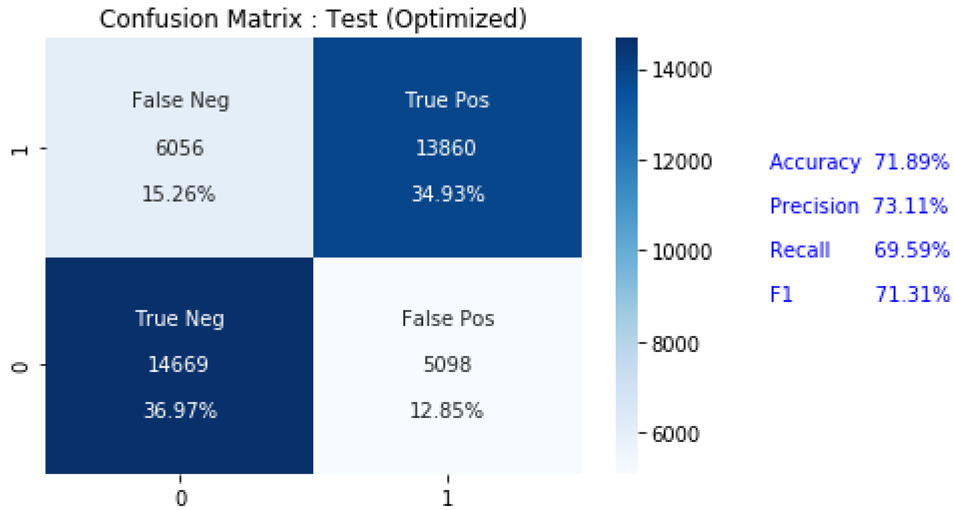
```
Training Data Accuracy = 0.9074490961628923  Test Data Accuracy = 0.718922460
4994582
```

In [72]:

```
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_train, y_trai
n))
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_validate, y_v
alidate))
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_test, y_test
))
```


Confusion Matrix : Test (Optimized)
False Neg
6698
5.63%
True Pos
53222
44.71%
True Neg
54808
46.04%
False Pos
4320
3.63%
0
1
50000
40000
30000
20000
10000
Accuracy 90.74%
Precision 92.49%
Recall 88.82%
F1 90.62%


Confusion Matrix : Test (Optimized)
False Neg
6131
15.45%
True Pos
13967
35.20%
True Neg
14513
36.57%
False Pos
5072
12.78%
0
1
14000
12000
10000
8000
6000
Accuracy 71.77%
Precision 73.36%
Recall 69.49%
F1 71.37%



Bagging Final Results

In [73]:

```
bagging_df = make_summary_table(result)
bagging_df
```

Out[73]:

	Settings	Data	Accuracy %	Precision %	Recall %	F1 %	Prediction Time (s)
0	Default	Train	96.96	97.51	96.41	96.96	107.39
1	Default	Validate	71.28	72.73	69.26	70.95	39.06
2	Default	Test	71.31	72.23	69.59	70.89	42.18
3	Optimized	Train	90.74	92.49	88.82	90.62	103.89
4	Optimized	Validate	71.77	73.36	69.49	71.37	37.63
5	Optimized	Test	71.89	73.11	69.59	71.31	34.34

In [ ]:

```

```

Naive Bayes (default)

In [74]:

```
nbayes = BernoulliNB()
pipeline = Pipeline([
    ('preprocessing', column_transformer),
    ('nbayes', nbayes)
], verbose=verbose)
```

In [75]:

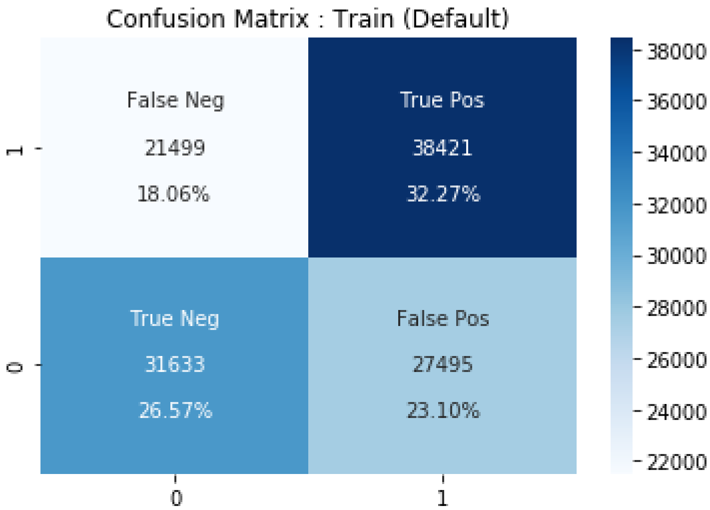
```
# Train
pipeline.fit(x_train, y_train)
# Accuracy
train_score = accuracy_score(y_train, pipeline.predict(x_train), normalize=True, sample_we
ight=None)
test_score = accuracy_score(y_test, pipeline.predict(x_test), normalize=True, sample_weigh
t=None)
train_score, test_score
```

Out[75]:

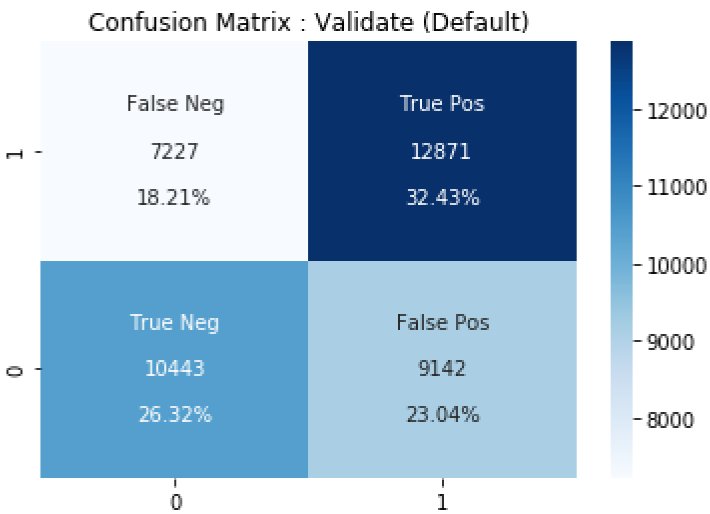
```
(0.5884517169545057, 0.5843056220547842)
```

In [76]:

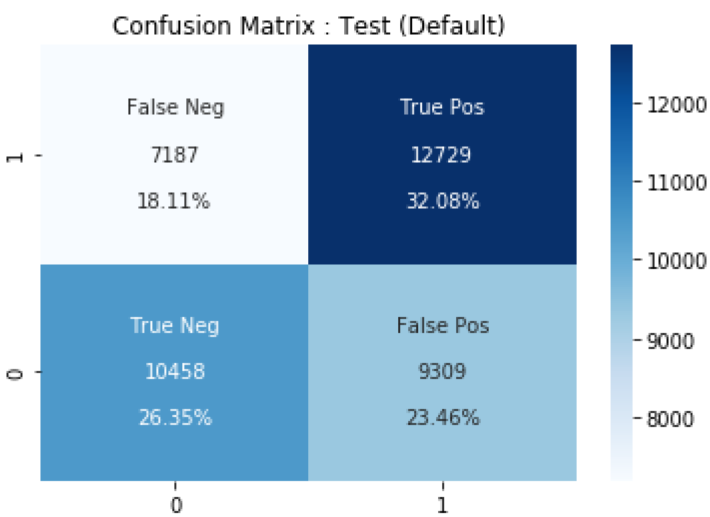
```
result = []
result.append(show_confusion_result('Confusion Matrix : Train (Default)', x_train, y_train
))
result.append(show_confusion_result('Confusion Matrix : Validate (Default)', x_validate, y
_validate))
result.append(show_confusion_result('Confusion Matrix : Test (Default)', x_test, y_test))
```



Accuracy 58.85%
Precision 58.29%
Recall 64.12%
F1 61.07%



Accuracy 58.75%
Precision 58.47%
Recall 64.04%
F1 61.13%



Accuracy 58.43%
Precision 57.76%
Recall 63.91%
F1 60.68%

Naive Bayes Hyper-Parameter Tuning

In [77]:

```
alpha = [0.5, 1.0, 1.5]
binarize = [0.5, 1.0, 1.5]
parameters = dict(nbayes__alpha = alpha,
                  nbayes__binarize = binarize)
clf = GridSearchCV(pipeline, parameters, cv=nfolds)
# Fit the grid search
clf.fit(x_validate, y_validate)
print('Best alpha:', clf.best_estimator_.get_params()['nbayes__alpha'])
print('Best binarize:', clf.best_estimator_.get_params()['nbayes__binarize'])
```

```
Best alpha: 1.0
Best binarize: 0.5
```

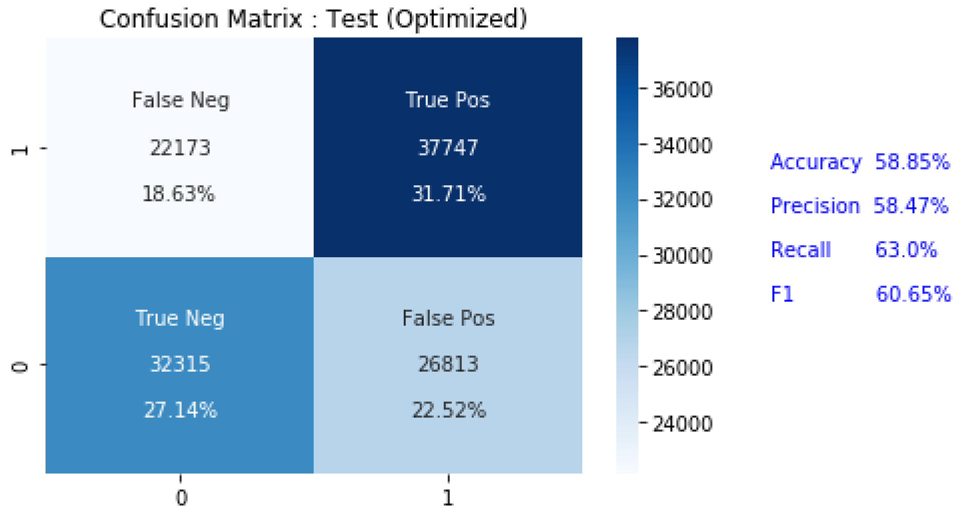
In [78]:

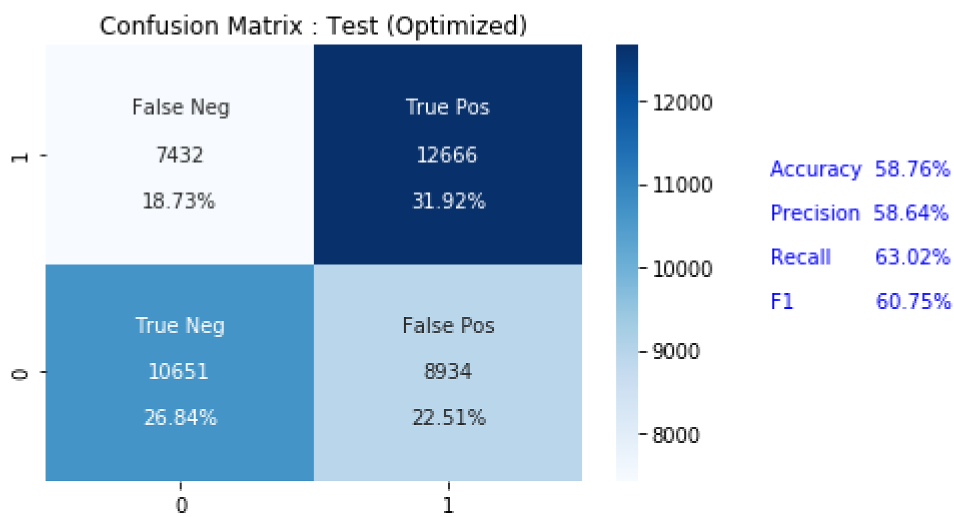
```
nbayes = clf.best_estimator_.get_params()['nbayes']
pipeline = Pipeline([
    ('preprocessing', column_transformer),
    ('nbayes', nbayes)
], verbose=verbose)
# Train
start = time.time();
pipeline.fit(x_train, y_train);
fit_time.append(time.time()-start);

# Accuracy
train_score = accuracy_score(y_train, pipeline.predict(x_train), normalize=True, sample_we
ight=None)
test_score = accuracy_score(y_test, pipeline.predict(x_test), normalize=True, sample_weigh
t=None)
print("Training Data Accuracy =", train_score, " Test Data Accuracy =", test_score)
```

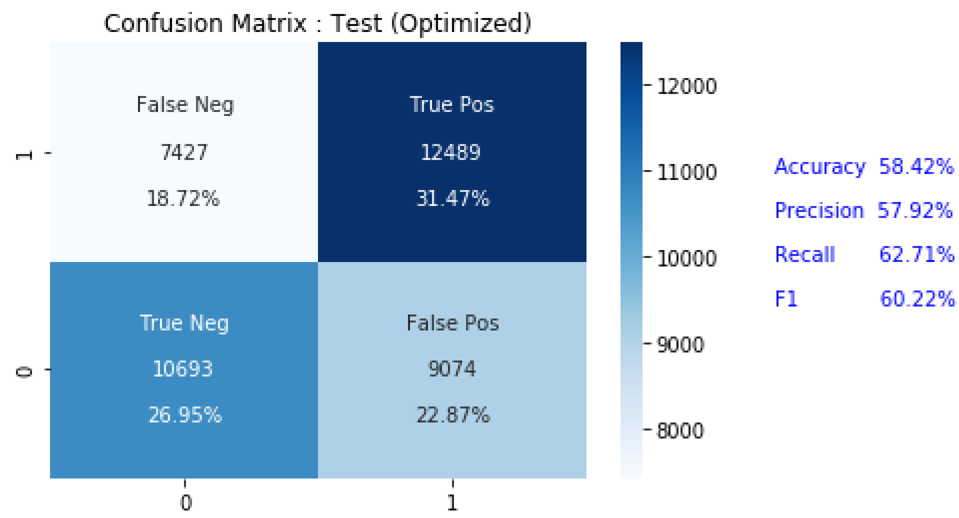
```
Training Data Accuracy = 0.5885189167394664  Test Data Accuracy = 0.584179623
5163672
```

In [79]:

```
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_train, y_trai
n))
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_validate, y_v
alidate))
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_test, y_test
))
```


Confusion Matrix : Test (Optimized)
False Neg
22173
18.63%
True Pos
37747
31.71%
True Neg
32315
27.14%
False Pos
26813
22.52%
1
0
0
1
36000
34000
32000
30000
28000
26000
24000
Accuracy 58.85%
Precision 58.47%
Recall 63.0%
F1 60.65%


Confusion Matrix : Test (Optimized)
False Neg
7432
18.73%
True Pos
12666
31.92%
True Neg
10651
26.84%
False Pos
8934
22.51%
1
0
0
1
12000
11000
10000
9000
8000
Accuracy 58.76%
Precision 58.64%
Recall 63.02%
F1 60.75%



Naive Bayes Final Results

In [80]:

```
nbayes_df = make_summary_table(result)
nbayes_df
```

Out[80]:

	Settings	Data	Accuracy %	Precision %	Recall %	F1 %	Prediction Time (s)
0	Default	Train	58.85	58.29	64.12	61.07	39.56
1	Default	Validate	58.75	58.47	64.04	61.13	14.06
2	Default	Test	58.43	57.76	63.91	60.68	14.06
3	Optimized	Train	58.85	58.47	63.00	60.65	43.24
4	Optimized	Validate	58.76	58.64	63.02	60.75	17.19
5	Optimized	Test	58.42	57.92	62.71	60.22	17.18

In [ ]:

```

```

Nearest Centroid (default)

In [81]:

```
ncentroid = NearestCentroid()
pipeline = Pipeline([
    ('preprocessing', column_transformer),
    ('ncentroid', ncentroid)
], verbose=verbose)
```

In [82]:

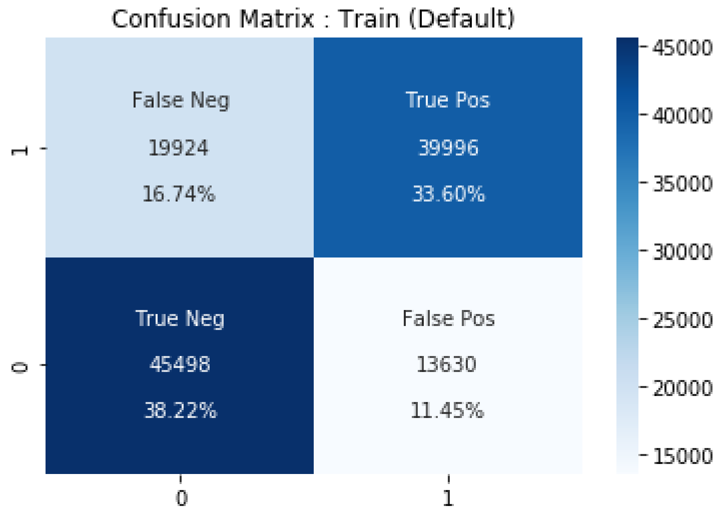
```
# Train
pipeline.fit(x_train, y_train)
# Accuracy
train_score = accuracy_score(y_train, pipeline.predict(x_train), normalize=True, sample_we
ight=None)
test_score = accuracy_score(y_test, pipeline.predict(x_test), normalize=True, sample_weigh
t=None)
train_score, test_score
```

Out[82]:

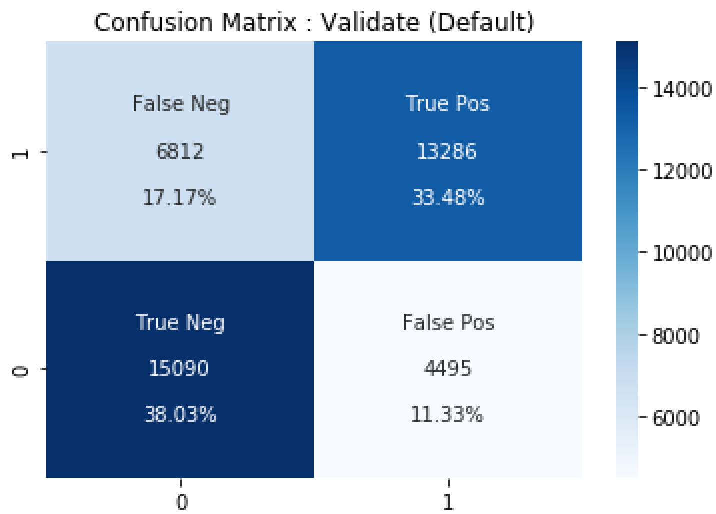
```
(0.7181473019286339, 0.7151425043469496)
```

In [83]:

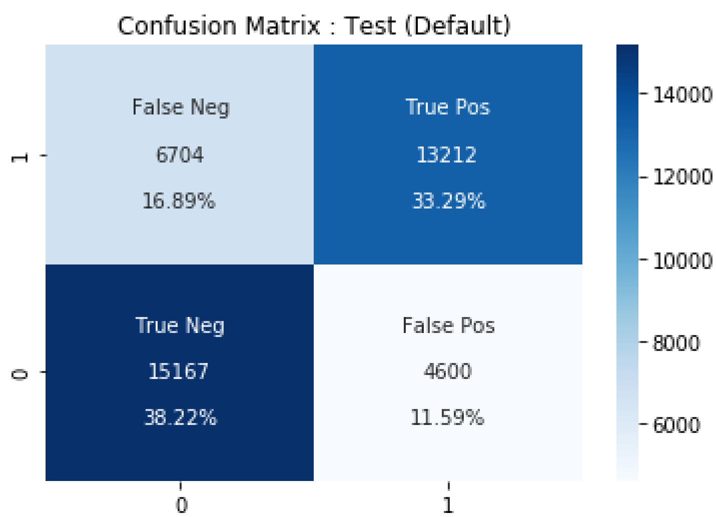
```
result = []
result.append(show_confusion_result('Confusion Matrix : Train (Default)', x_train, y_train
))
result.append(show_confusion_result('Confusion Matrix : Validate (Default)', x_validate, y
_validate))
result.append(show_confusion_result('Confusion Matrix : Test (Default)', x_test, y_test))
```



Accuracy 71.81%
Precision 74.58%
Recall 66.75%
F1 70.45%



Accuracy 71.51%
Precision 74.72%
Recall 66.11%
F1 70.15%



Accuracy 71.51%
Precision 74.17%
Recall 66.34%
F1 70.04%

Nearest Centroid Hyper-Parameter Tuning

In [84]:

```
metric = ['euclidean', 'manhattan']
parameters = dict(ncentroid__metric = metric)
clf = GridSearchCV(pipeline, parameters, cv=nfolds)
# Fit the grid search
clf.fit(x_validate, y_validate)
print('Best metric:', clf.best_estimator_.get_params()['ncentroid__metric'])
```

```
Best metric: euclidean
```

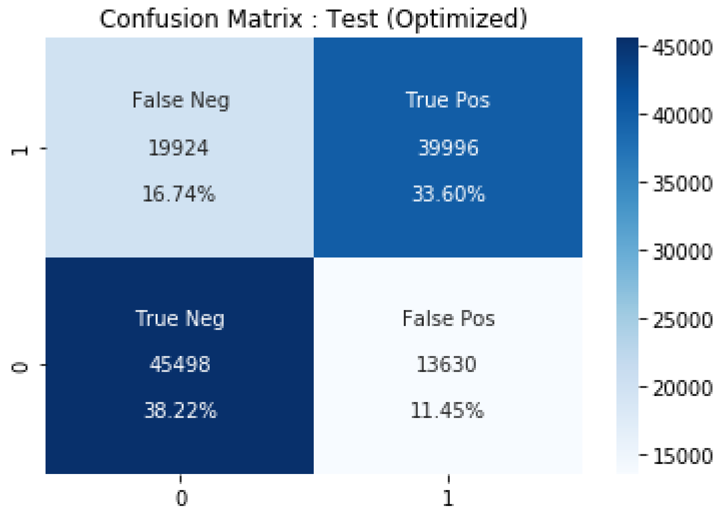
In [85]:

```
ncentroid = clf.best_estimator_.get_params()['ncentroid']
pipeline = Pipeline([
    ('preprocessing', column_transformer),
    ('ncentroid', ncentroid)
], verbose=verbose)
# Train
start = time.time();
pipeline.fit(x_train, y_train);
fit_time.append(time.time()-start);

# Accuracy
train_score = accuracy_score(y_train, pipeline.predict(x_train), normalize=True, sample_we
ight=None)
test_score = accuracy_score(y_test, pipeline.predict(x_test), normalize=True, sample_weigh
t=None)
print("Training Data Accuracy =", train_score, " Test Data Accuracy =", test_score)
```

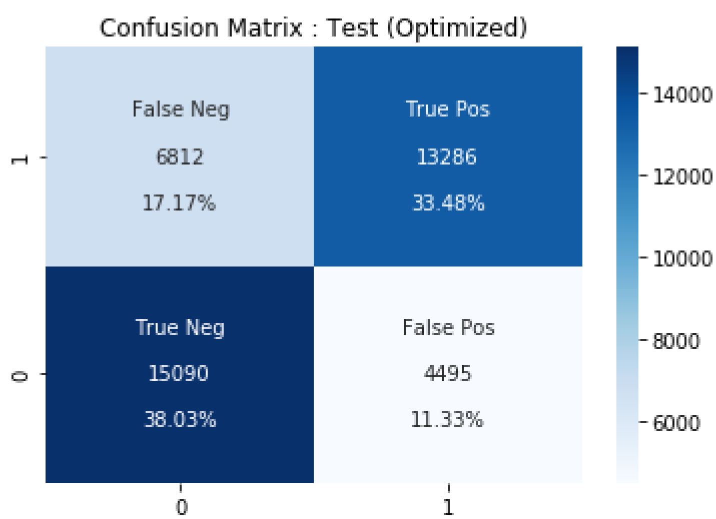
```
Training Data Accuracy = 0.7181473019286339  Test Data Accuracy = 0.715142504
3469496
```

In [86]:

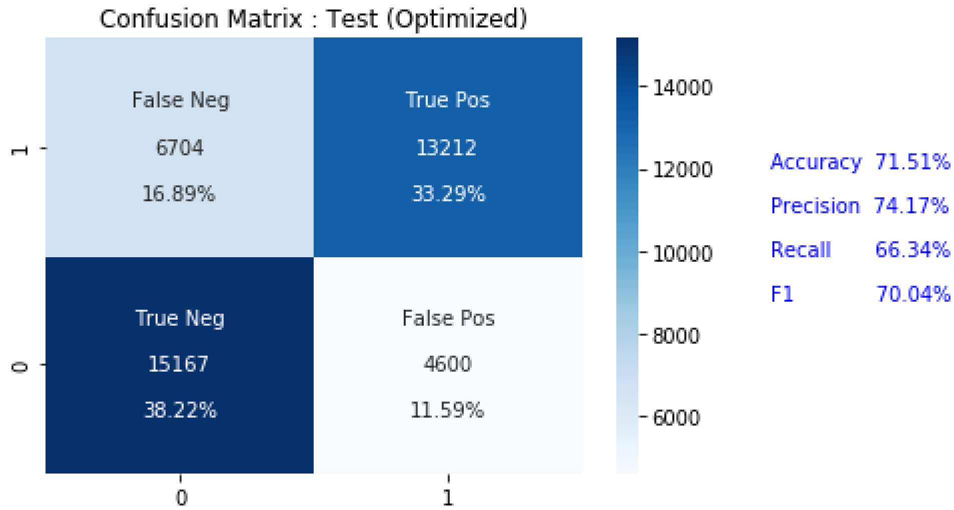
```
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_train, y_trai
n))
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_validate, y_v
alidate))
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_test, y_test
))
```



Accuracy	71.81%
Precision	74.58%
Recall	66.75%
F1	70.45%



Accuracy	71.51%
Precision	74.72%
Recall	66.11%
F1	70.15%



Nearest Centroid Final Results

In [87]:

```
ncentroid_df = make_summary_table(result)
ncentroid_df
```

Out[87]:

	Settings	Data	Accuracy %	Precision %	Recall %	F1 %	Prediction Time (s)
0	Default	Train	71.81	74.58	66.75	70.45	36.77
1	Default	Validate	71.51	74.72	66.11	70.15	15.62
2	Default	Test	71.51	74.17	66.34	70.04	14.06
3	Optimized	Train	71.81	74.58	66.75	70.45	38.11
4	Optimized	Validate	71.51	74.72	66.11	70.15	15.74
5	Optimized	Test	71.51	74.17	66.34	70.04	15.62

In [ ]:

```

```

Multi Layer Perceptron (default)

In [88]:

```
mlp = MLPClassifier()
pipeline = Pipeline([
    ('preprocessing', column_transformer),
    ('mlp', mlp)
], verbose=verbose)
```

In [89]:

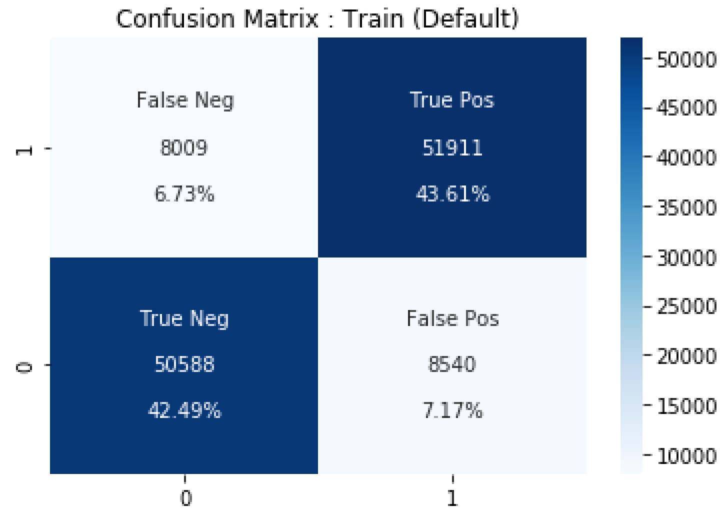
```
# Train
pipeline.fit(x_train, y_train)
# Accuracy
train_score = accuracy_score(y_train, pipeline.predict(x_train), normalize=True, sample_we
ight=None)
test_score = accuracy_score(y_test, pipeline.predict(x_test), normalize=True, sample_weigh
t=None)
train_score, test_score
```

Out[89]:

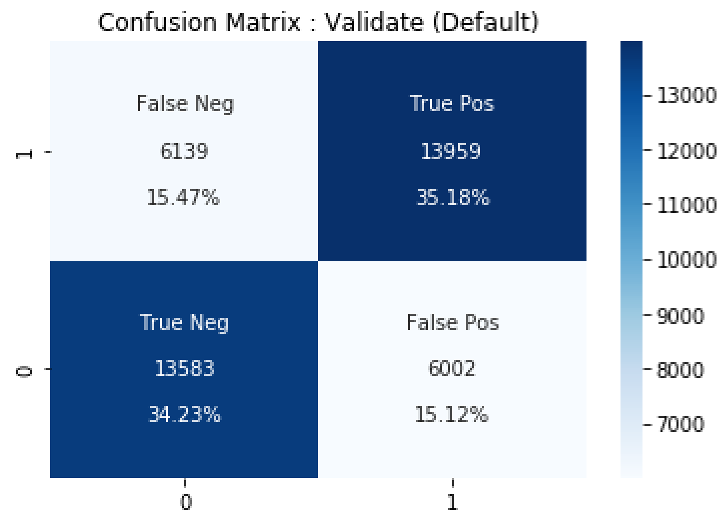
```
(0.8609888448356965, 0.6982587001990777)
```

In [90]:

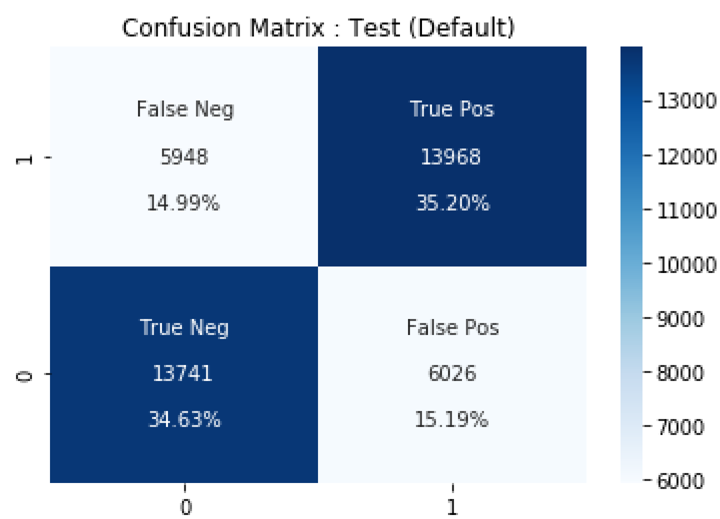
```
result = []
result.append(show_confusion_result('Confusion Matrix : Train (Default)', x_train, y_train
))
result.append(show_confusion_result('Confusion Matrix : Validate (Default)', x_validate, y
_validate))
result.append(show_confusion_result('Confusion Matrix : Test (Default)', x_test, y_test))
```



Accuracy 86.1%
Precision 85.87%
Recall 86.63%
F1 86.25%



Accuracy 69.41%
Precision 69.93%
Recall 69.45%
F1 69.69%



Accuracy 69.83%
Precision 69.86%
Recall 70.13%
F1 70.0%

Multi Layer Perceptron Hyper-Parameter Tuning

In [91]:

```
nfolds = 2 # too slow for too many folds
activation = ['logistic', 'tanh', 'relu']
solver = ['lbfgs', 'adam']

parameters = dict(mlp__activation = activation,
                  mlp__solver = solver)
clf = GridSearchCV(pipeline, parameters, cv=nfolds)
# Fit the grid search
clf.fit(x_validate, y_validate)
print('Best activation:', clf.best_estimator_.get_params()['mlp__activation'])
print('Best solver:', clf.best_estimator_.get_params()['mlp__solver'])
```

```
Best activation: logistic
Best solver: lbfgs
```

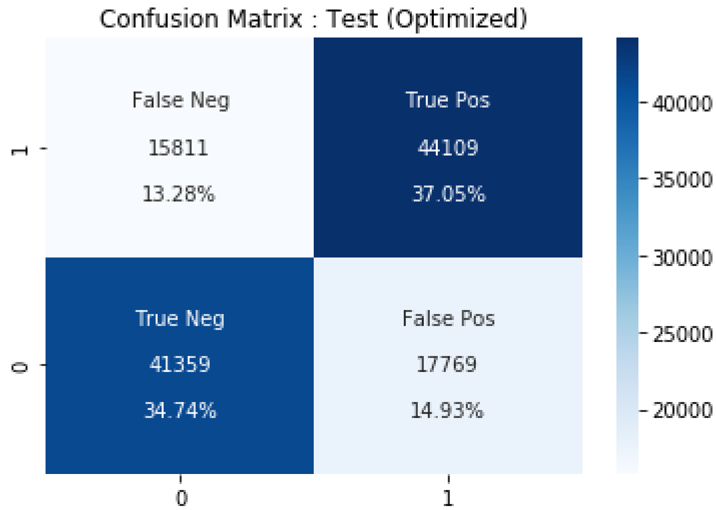
In [92]:

```
mlp = clf.best_estimator_.get_params()['mlp']
pipeline = Pipeline([
    ('preprocessing', column_transformer),
    ('mlp', mlp)
], verbose=verbose)
# Train
start = time.time();
pipeline.fit(x_train, y_train);
fit_time.append(time.time()-start);

# Accuracy
train_score = accuracy_score(y_train, pipeline.predict(x_train), normalize=True, sample_we
ight=None)
test_score = accuracy_score(y_test, pipeline.predict(x_test), normalize=True, sample_weigh
t=None)
print("Training Data Accuracy =", train_score, " Test Data Accuracy =", test_score)
```

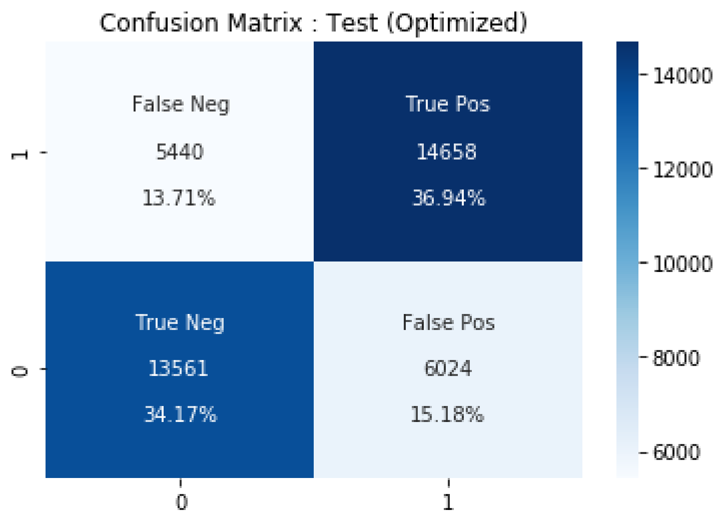
```
Training Data Accuracy = 0.7179289026275116  Test Data Accuracy = 0.713151727
4399617
```

In [93]:

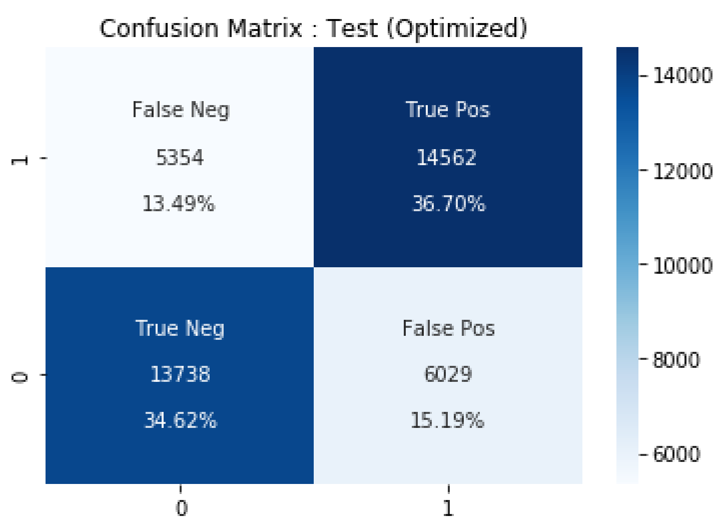
```
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_train, y_trai
n))
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_validate, y_v
alidate))
result.append(show_confusion_result('Confusion Matrix : Test (Optimized)', x_test, y_test
))
```



Accuracy 71.79%
Precision 71.28%
Recall 73.61%
F1 72.43%



Accuracy 71.11%
Precision 70.87%
Recall 72.93%
F1 71.89%



Accuracy 71.32%
Precision 70.72%
Recall 73.12%
F1 71.9%

Multi Layer Perceptron Final Results

In [94]:

```
mlp_df = make_summary_table(result)
mlp_df
```

Out[94]:

	Settings	Data	Accuracy %	Precision %	Recall %	F1 %	Prediction Time (s)
0	Default	Train	86.10	85.87	86.63	86.25	143.55
1	Default	Validate	69.41	69.93	69.45	69.69	48.57
2	Default	Test	69.83	69.86	70.13	70.00	48.54
3	Optimized	Train	71.79	71.28	73.61	72.43	51.69
4	Optimized	Validate	71.11	70.87	72.93	71.89	17.19
5	Optimized	Test	71.32	70.72	73.12	71.90	18.75

Model Final Results

In [95]:

```
summary_df = decisiontree_df.assign(Model="Decision Tree").assign(Fit_Time=fit_time[0])
summary_df = summary_df.append(randomforest_df.assign(Model="Random Forest").assign(Fit_Ti
me=fit_time[1]))
summary_df = summary_df.append(svc_df.assign(Model="SVC").assign(Fit_Time=fit_time[2]))
summary_df = summary_df.append(gboost_df.assign(Model="Gradient Boosting").assign(Fit_Time
=fit_time[3]))
summary_df = summary_df.append(bagging_df.assign(Model="Bagging").assign(Fit_Time=fit_time
[4]))
summary_df = summary_df.append(nbayes_df.assign(Model="Naive Bayes (Bernoulli)").assign(Fi
t_Time=fit_time[5]))
summary_df = summary_df.append(ncentroid_df.assign(Model="Centroid").assign(Fit_Time=fit_t
ime[6]))
summary_df = summary_df.append(mlp_df.assign(Model="Multi Layer Perceptron").assign(Fit_Ti
me=fit_time[7]))

summary_df = summary_df[summary_df['Settings']=='Optimized'].drop('Settings', axis=1).rese
t_index(drop=True)[["Model", "Data", "Accuracy %", "Precision %", "Recall %", "F1 %", "Pre
diction Time (s)", "Fit_Time"]].rename({"Fit_Time":"Train Time (s)"}, axis=1)
summary_df
```

Out[95]:

	Model	Data	Accuracy %	Precision %	Recall %	F1 %	Prediction Time (s)	Train Time (s)
0	Decision Tree	Train	72.02	74.65	67.25	70.75	37.84	1.041455
1	Decision Tree	Validate	71.71	74.77	66.61	70.45	14.69	1.041455
2	Decision Tree	Test	71.66	74.18	66.76	70.28	15.62	1.041455
3	Random Forest	Train	86.24	86.22	86.49	86.36	97.95	29.303271
4	Random Forest	Validate	74.58	75.27	74.19	74.72	38.36	29.303271
5	Random Forest	Test	74.60	74.88	74.32	74.60	37.50	29.303271
6	SVC	Train	62.15	68.22	46.45	55.27	54.91	0.780528
7	SVC	Validate	62.14	68.57	46.61	55.49	22.26	0.780528
8	SVC	Test	62.06	67.77	46.55	55.19	25.12	0.780528
9	Gradient Boosting	Train	73.14	73.55	72.83	73.19	91.03	53.543317
10	Gradient Boosting	Validate	72.68	73.43	72.17	72.79	31.50	53.543317
11	Gradient Boosting	Test	72.69	73.06	72.22	72.64	26.56	53.543317
12	Bagging	Train	90.74	92.49	88.82	90.62	103.89	114.754657
13	Bagging	Validate	71.77	73.36	69.49	71.37	37.63	114.754657
14	Bagging	Test	71.89	73.11	69.59	71.31	34.34	114.754657
15	Naive Bayes (Bernoulli)	Train	58.85	58.47	63.00	60.65	43.24	0.478188
16	Naive Bayes (Bernoulli)	Validate	58.76	58.64	63.02	60.75	17.19	0.478188
17	Naive Bayes (Bernoulli)	Test	58.42	57.92	62.71	60.22	17.18	0.478188
18	Centroid	Train	71.81	74.58	66.75	70.45	38.11	0.432970
19	Centroid	Validate	71.51	74.72	66.11	70.15	15.74	0.432970
20	Centroid	Test	71.51	74.17	66.34	70.04	15.62	0.432970
21	Multi Layer Perceptron	Train	71.79	71.28	73.61	72.43	51.69	115.882649
22	Multi Layer Perceptron	Validate	71.11	70.87	72.93	71.89	17.19	115.882649
23	Multi Layer Perceptron	Test	71.32	70.72	73.12	71.90	18.75	115.882649

In [ ]:

In [ ]:

```

```

In [ ]:

```

```