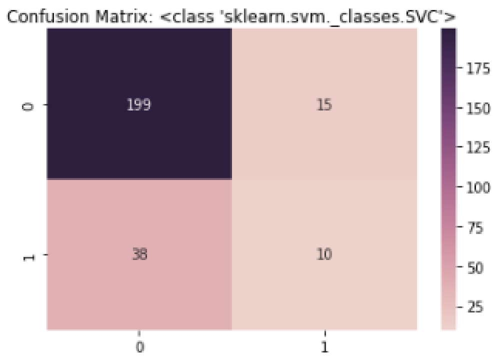
condition	0.84	0.93	0.88	214
stroke	0.40	0.21	0.27	48
accuracy			0.80	262
macro avg	0.62	0.57	0.58	262
weighted avg	0.76	0.80	0.77	262

In [244]:
```
#testing
cf_matrix = confusion_matrix(y_test, model.predict(X_test))
plt.title('Confusion Matrix: {}'.format(SVC))
sns.heatmap(cf_matrix, annot = True, fmt = 'g', cmap = sns.cubehelix_palette(as_cmap=True))
plt.show()
#dont reduce to much
#printed format with 4 numbers
#confusion matrix for training and end result
```



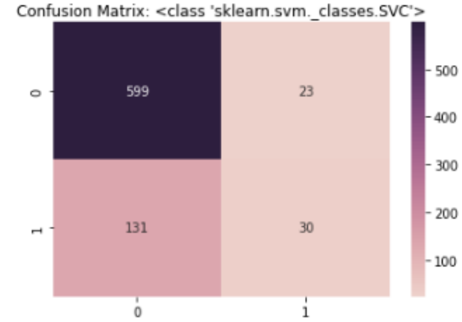
In [245]:
```
y_true = y_test
y_pred = model.predict(X_test)
confusion_matrix(y_true, y_pred)
```

Out[245]:
```
array([[199,  15],
       [ 38,  10]], dtype=int64)
```

In [246]:
```
tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
accuracy = (tp + tn) / (tp + fp + tn + fn)
print(f"Accuracy: {accuracy}")
print('true negative', tn, '\n',
      'false positive', fp, '\n',
      'false negative', fn, '\n',
      'true positive', tp, '\n')
```

```
Accuracy: 0.7977099236641222
true negative 199
 false positive 15
 false negative 38
 true positive 10
```

In [247]:
```
#raining
cf_matrix = confusion_matrix(y_train, model.predict(X_train))
plt.title('Confusion Matrix: {}'.format(SVC))
sns.heatmap(cf_matrix, annot = True, fmt = 'g', cmap = sns.cubehelix_palette(as_cmap=True))
plt.show()
```



In [248]:
```
y_true = y_train
y_pred = model.predict(X_train)
confusion_matrix(y_true, y_pred)
```

Out[248]:
```
array([[599,  23],
       [131,  30]], dtype=int64)
```

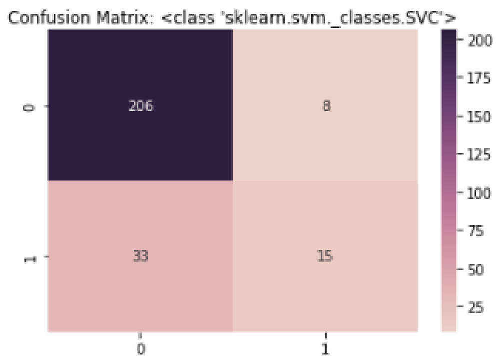
In [249]:
```
tn, fp, fn, tp = confusion_matrix(y_train, y_pred).ravel()
accuracy = (tp + tn) / (tp + fp + tn + fn)
print(f"Accuracy: {accuracy}")
print('true negative', tn, '\n',
      'false positive', fp, '\n',
      'false negative', fn, '\n',
      'true positive', tp, '\n')
```

```
Accuracy: 0.8033205619412516
true negative 599
 false positive 23
 false negative 131
 true positive 30
```

SVM using the linear method and under sampling I managed to produce the following machine learning model. Out of the three final models I made this one was the second best out of the bunch. It much better than sigmoid because it can tell more of the false cases and reduces the number of cases that are false negative. This means less cases that had strokes aren't be misclassified as not having them. Also a few more cases are classified as true positive. Also a few less cases that are false positive so not telling people unnecessarily worried and misclassified about having stroke.

	precision	recall	f1-score	support
condition	0.86	0.96	0.91	214
stroke	0.65	0.31	0.42	48
accuracy			0.84	262
macro avg	0.76	0.64	0.67	262
weighted avg	0.82	0.84	0.82	262

In [253]:
```
#testing
cf_matrix = confusion_matrix(y_test, model.predict(X_test))
plt.title('Confusion Matrix: {}'.format(SVC))
sns.heatmap(cf_matrix, annot = True, fmt = 'g', cmap = sns.cubehelix_palette(as_cmap=True))
plt.show()
```



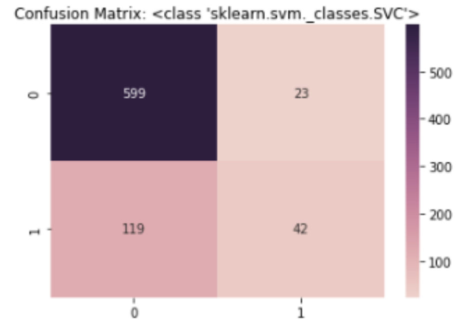
In [254]:
```
y_true = y_test
y_pred = model.predict(X_test)
confusion_matrix(y_true, y_pred)
```

Out[254]:
```
array([[206,   8],
       [ 33,  15]], dtype=int64)
```

In [255]:
```
tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
accuracy = (tp + tn) / (tp + fp + tn + fn)
print(f"Accuracy: {accuracy}")
print('true negative', tn, '\n',
      'false positive', fp, '\n',
      'false negative', fn, '\n',
      'true positive', tp, '\n')
```

```
Accuracy: 0.8435114503816794
true negative 206
 false positive 8
 false negative 33
 true positive 15
```

In [256]:
```
#training
cf_matrix = confusion_matrix(y_train, model.predict(X_train))
plt.title('Confusion Matrix: {}'.format(SVC))
sns.heatmap(cf_matrix, annot = True, fmt = 'g', cmap = sns.cubehelix_palette(as_cmap=True))
plt.show()
```



In [257]:
```
y_true = y_train
y_pred = model.predict(X_train)
confusion_matrix(y_true, y_pred)
```

Out[257]:
```
array([[599,  23],
       [119,  42]], dtype=int64)
```

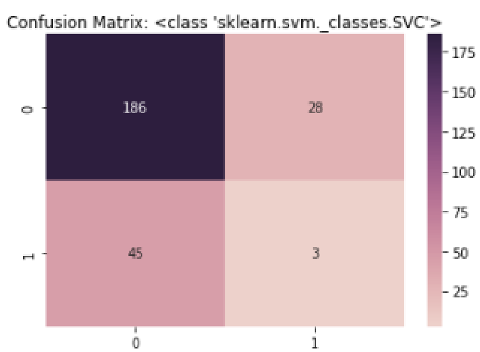
In [258]:
```
tn, fp, fn, tp = confusion_matrix(y_train, y_pred).ravel()
accuracy = (tp + tn) / (tp + fp + tn + fn)
print(f"Accuracy: {accuracy}")
print('true negative', tn, '\n',
      'false positive', fp, '\n',
      'false negative', fn, '\n',
      'true positive', tp, '\n')
```

```
Accuracy: 0.8186462324393359
true negative 599
 false positive 23
 false negative 119
 true positive 42
```

SVM using the poly method and under sampling I managed to produce the following machine learning model. This was the best model I was able to make as it can identify more true positive cases meaning it can tell more of the people who had strokes from those that didn't. It also has the lowest number of false negative cases so it's the least dangerous model as it will miss less people who did have a stroke. The number of false positive cases is also the lowest meaning it won't misclassify to many people as being at risk. However none of my models are good enough to actually be used as they miss far too many people who have strokes.

In [259]:
```python
# Support vector machine sigmoid classifier
from sklearn.svm import SVC
model = SVC(kernel='sigmoid')
model.fit(X_train, y_train)
```
Out[259]:
```
SVC(kernel='sigmoid')
```
In [260]:
```python
# Model Accuracy
print('Test Acc: %.3f' % model.score(X_test, y_test))
```
```
Test Acc: 0.721
```
In [261]:
```python
# Calculate the classification report
from sklearn.metrics import classification_report
predictions = model.predict(X_test)
print(classification_report(y_test, predictions,
                            target_names=target_names))
```
```
              precision    recall  f1-score   support

   condition       0.81      0.87      0.84       214
      stroke       0.10      0.06      0.08        48

    accuracy                           0.72       262
   macro avg       0.45      0.47      0.46       262
weighted avg       0.68      0.72      0.70       262
```
In [262]:
```python
#testing
cf_matrix = confusion_matrix(y_test, model.predict(X_test))
plt.title('Confusion Matrix: {}'.format(SVC))
sns.heatmap(cf_matrix, annot = True, fmt = 'g', cmap = sns.cubehelix_palette(as_cmap=True))
plt.show()
```


SVM using sigmoid method and under sampling I managed to produce the following machine learning model. Which was a minor imporvement over the model without the under sampling however it was barely better. It was wose than the other two models which was why iy was dropped and received no further updates to it's code. It did a terrible job of identifying stroke cases and in fact classified significantly more stroke cases as not having a stroke. This makes the model the most dangerous and useless of the three. It doesn't identify false cases particually well. In other words the acurracy of the model is much worse than the other two especially in the areas that count.

Out[222]:

	age	hypertension	heart_disease	avg_glucose_level	bmi
0	67.0	0	1	228.69	36.6
1	80.0	0	1	105.92	32.5
2	49.0	0	0	171.23	34.4
3	79.0	1	0	174.12	24.0
4	81.0	0	0	186.21	29.0

In [223]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data, target, random_state=42)
```

In [224]:
```
# Support vector machine linear classifier
from sklearn.svm import SVC
model = SVC(kernel='linear')
model.fit(X_train, y_train)
```

Out[224]:
```
SVC(kernel='linear')
```

In [225]:
```
# Model Accuracy
print('Test Acc: %.3f' % model.score(X_test, y_test))
```

```
Test Acc: 0.949
```

In [226]:
```
# Calculate the classification report
from sklearn.metrics import classification_report
predictions = model.predict(X_test)
print(classification_report(y_test, predictions,
                            target_names=target_names))
```

```
              precision    recall  f1-score   support

   condition       0.95      1.00      0.97      1164
      stroke       0.00      0.00      0.00        63

    accuracy                           0.95      1227
   macro avg       0.47      0.50      0.49      1227
weighted avg       0.90      0.95      0.92      1227
```

```
C:\Users\marcus garnham\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision
and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\marcus garnham\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision
and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\marcus garnham\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision
and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

This is an example of the models produced by Linear without using under sampling. The sigmoid and poly models also porduced the same results without under sampling. There was no ability of the models to identify strokes. Which makes the model completely useless.

In [233]:
```
strokes = len(df[df['stroke'] == 1])
print(strokes)
```

209

In [234]:
```
df_strokes = df[df['stroke'] == 1 ]
df_strokes
```

Out[234]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
2	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
3	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1
4	Male	81.0	0	0	Yes	Private	Urban	186.21	29.0	formerly smoked	1
...	...	...	...	...	...	...	...	...	...	...	...
204	Female	68.0	1	1	Yes	Private	Urban	247.51	40.5	formerly smoked	1
205	Male	57.0	0	0	Yes	Private	Rural	84.96	36.7	Unknown	1
206	Female	14.0	0	0	No	children	Rural	57.93	30.9	Unknown	1
207	Female	75.0	0	0	Yes	Self-employed	Rural	78.80	29.3	formerly smoked	1
208	Female	78.0	0	0	Yes	Private	Rural	78.81	19.6	Unknown	1

209 rows × 11 columns

In [235]:
```
#no_strokes = df[df.stroke == 0].index
#print(no_strokes)
no_strokes = df[df['stroke'] == 0 ]
no_strokes
```

Out[235]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
209	Male	3.0	0	0	No	children	Rural	95.12	18.0	Unknown	0
210	Male	58.0	1	0	Yes	Private	Urban	87.96	39.2	never smoked	0
211	Female	8.0	0	0	No	Private	Urban	110.89	17.6	Unknown	0
212	Female	70.0	0	0	Yes	Private	Rural	69.04	35.9	formerly smoked	0
213	Male	14.0	0	0	No	Never_worked	Rural	161.28	19.1	Unknown	0
...	...	...	...	...	...	...	...	...	...	...	...
4903	Female	13.0	0	0	No	children	Rural	103.08	18.6	Unknown	0
4904	Female	81.0	0	0	Yes	Self-employed	Urban	125.20	40.0	never smoked	0
4905	Female	35.0	0	0	Yes	Self-employed	Rural	82.99	30.6	never smoked	0
4906	Male	51.0	0	0	Yes	Private	Rural	166.29	25.6	formerly smoked	0
4907	Female	44.0	0	0	Yes	Govt_job	Urban	85.28	26.2	Unknown	0

4699 rows × 11 columns

Here is under sampling code and how I produced it using only pandas. The code is robust and can be reused as the number of none stroke cases is dependent on number of stroke cases and that is calculated by calculating the number of them and putting them in a variable. The two new data frames are then concatonated together in order to make a new data set. Under sampling has an advantage over oversampling in this case as it's not making up patient data which might not be acurrate.

In [236]:
```
no_strokes_2 = no_strokes.sample(n=strokes*4, replace=False)
no_strokes_2
```

Out[236]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
4858	Female	49.0	0	0	Yes	Govt_job	Urban	69.92	47.6	never smoked	0
4799	Female	40.0	0	0	Yes	Private	Urban	93.97	23.6	never smoked	0
986	Female	79.0	0	0	Yes	Govt_job	Urban	93.89	30.4	never smoked	0
3294	Male	62.0	0	0	Yes	Private	Rural	60.39	26.9	Unknown	0
3442	Female	36.0	0	0	Yes	Private	Rural	71.32	43.9	smokes	0
...	...	...	...	...	...	...	...	...	...	...	...
2434	Female	28.0	0	0	Yes	Private	Rural	94.15	23.1	smokes	0
1963	Female	66.0	0	0	Yes	Private	Urban	202.05	31.7	smokes	0
4758	Female	81.0	0	0	No	Self-employed	Urban	57.42	33.7	never smoked	0
1070	Female	47.0	0	0	Yes	Private	Rural	195.04	45.5	never smoked	0
2323	Female	73.0	0	0	Yes	Self-employed	Urban	87.56	24.1	never smoked	0

836 rows × 11 columns

In [237]:
```
Undersample_concat = pd.concat([no_strokes_2, df_strokes])
Undersample_concat
```

Out[237]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
4858	Female	49.0	0	0	Yes	Govt_job	Urban	69.92	47.6	never smoked	0
4799	Female	40.0	0	0	Yes	Private	Urban	93.97	23.6	never smoked	0
986	Female	79.0	0	0	Yes	Govt_job	Urban	93.89	30.4	never smoked	0
3294	Male	62.0	0	0	Yes	Private	Rural	60.39	26.9	Unknown	0
3442	Female	36.0	0	0	Yes	Private	Rural	71.32	43.9	smokes	0
...	...	...	...	...	...	...	...	...	...	...	...
204	Female	68.0	1	1	Yes	Private	Urban	247.51	40.5	formerly smoked	1
205	Male	57.0	0	0	Yes	Private	Rural	84.96	36.7	Unknown	1
206	Female	14.0	0	0	No	children	Rural	57.93	30.9	Unknown	1
207	Female	75.0	0	0	Yes	Self-employed	Rural	78.80	29.3	formerly smoked	1
208	Female	78.0	0	0	Yes	Private	Rural	78.81	19.6	Unknown	1

1045 rows × 11 columns

In [238]:
```
target2 = Undersample_concat["stroke"]
target_names2 = ["condition", "stroke"]
```

In [239]:
```
data2 = Undersample_concat.drop(["stroke","gender","ever_married","work_type","Residence_type","smoking_status"], axis=1)
feature_names2 = data2.columns
data2
```

Out[239]:

	age	hypertension	heart_disease	avg_glucose_level	bmi
4858	49.0	0	0	69.92	47.6
4799	40.0	0	0	93.97	23.6
986	79.0	0	0	93.89	30.4
3294	62.0	0	0	60.39	26.9
3442	36.0	0	0	71.32	43.9
...	...	...	...	...	...
204	68.0	1	1	247.51	40.5
205	57.0	0	0	84.96	36.7
206	14.0	0	0	57.93	30.9
207	75.0	0	0	78.80	29.3
208	78.0	0	0	78.81	19.6

1045 rows × 5 columns