Modeling

In [1]:

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

# sklearn models
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier

# sklearn preprocessing
from sklearn.preprocessing import StandardScaler, FunctionTransformer
from sklearn.compose import ColumnTransformer,  make_column_selector as selecto
from sklearn.pipeline import Pipeline

# sklearn metrics and validation
from sklearn.model_selection import cross_val_score, train_test_split, GridSear
from sklearn.metrics import accuracy_score, precision_score, \
f1_score, plot_confusion_matrix, plot_roc_curve, recall_score, \
classification_report, roc_auc_score, make_scorer

# xgboost
import xgboost
```

In [2]:

```
# Loading the dataset
df = pd.read_csv('./Data/syriatel_clean.csv', index_col=0)
```

In [3]:

```
df.head()
```

Out[3]:

	state	account length	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	...
0	KS	128	0	1	25	265.1	110	45.07	197.4	99	...
1	OH	107	0	1	26	161.6	123	27.47	195.5	103	...
2	NJ	137	0	0	0	243.4	114	41.38	121.2	110	...
3	OH	84	1	0	0	299.4	71	50.90	61.9	88	...
4	OK	75	1	0	0	166.7	113	28.34	148.3	122	...

5 rows × 22 columns

Business Understanding

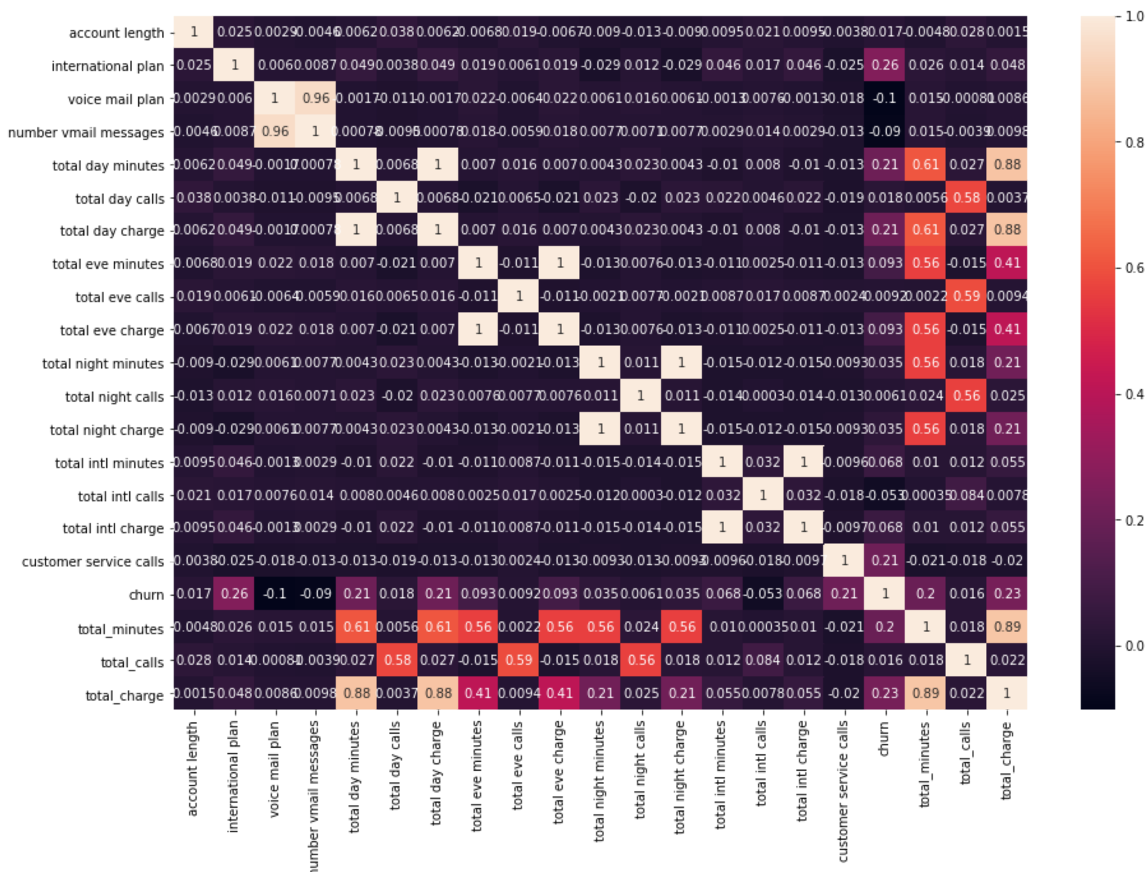
For this business task, the company wants to use this model to predict whether a customer will churn or not. Once they identify customers who are likely to churn they are going to reach out to them and provide them with incentives to stay with the company. For this reason, the company wants to minimize False Negatives as they do not want to lose out on customers who are about to churn. Customer acquisition costs much higher than customer retention so they do not want customers who are about to churn to go unnoticed. For this reason, I am going to focus on models that have high recall scores as they are impacted by False Negatives.

Correlation

In [4]:

```
# fig ax set up
fig, ax = plt.subplots(figsize=(15,10))

# heatmatp
sns.heatmap(data=df.corr(), annot=True)
```

Out[4]: <AxesSubplot:>



I am going to get rid of the following columns for right due to mulitcollinearity:

anything column with charge, total_minutes, total_calls, total_charge, number of voicemail messages

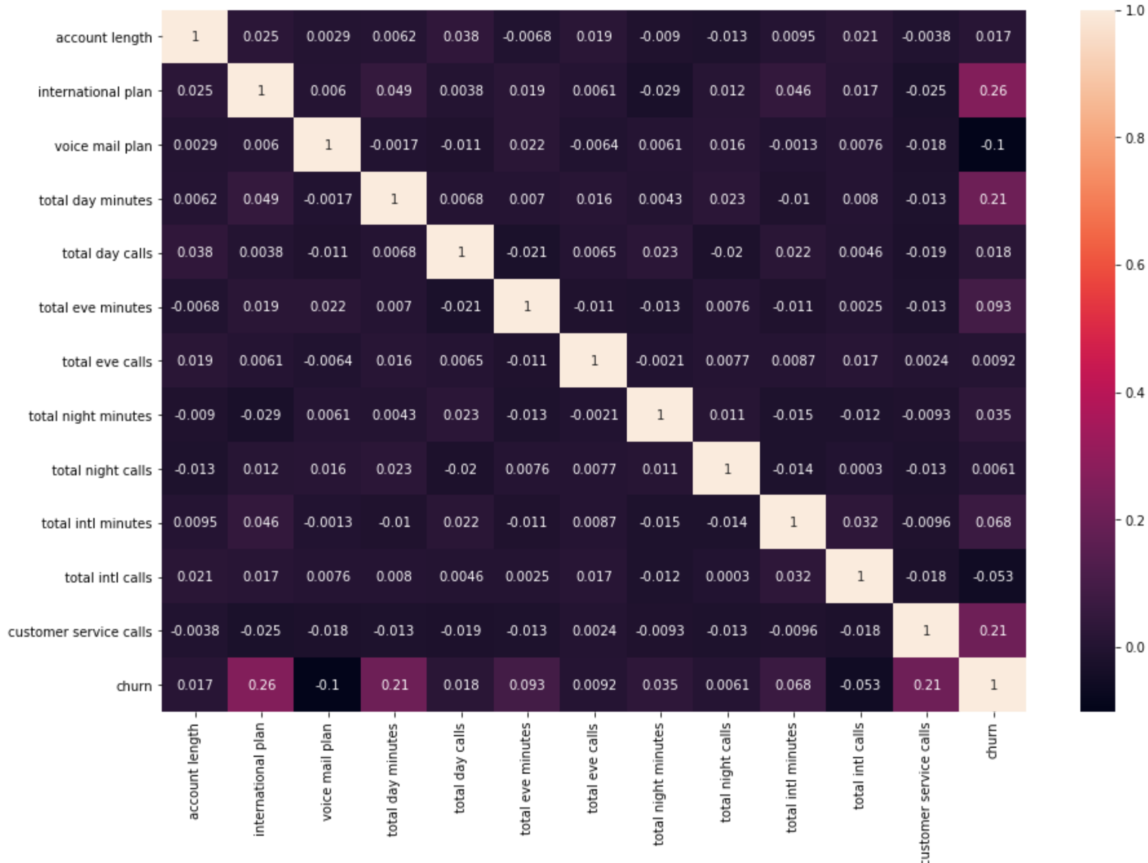
In [5]:

```python
cols_to_drop = ['number vmail messages', 'total day charge', 'total eve charge'
                'total night charge', 'total_minutes', 'total_calls', 'total_cha
                'total intl charge']

# new df
df2 = df.drop(columns=cols_to_drop)
```

In [6]:

```python
# fig ax set up
fig, ax = plt.subplots(figsize=(15,10))

# new heatmap
sns.heatmap(data=df2.corr(), annot=True)
```

Out[6]: <AxesSubplot:>



Train Test Split

In [7]:

```python
# split data frame
X = df2.drop(columns=['churn', 'state'])
y = df2['churn']

# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42,
                                                    test_size=0.3, stratify=y)
```

Helper Function

In [8]:
```python
# function for printing out metrics and plots
def print_metrics(estimator, X_test, y_test):

    # plot confusiont matrix
    plot_confusion_matrix(estimator,
                          X=X_test,
                          y_true=y_test)

    # metrics
    y_pred = estimator.predict(X_test)

    print('Accuracy score:', round(accuracy_score(y_test, y_pred), 3))
    print('Precision score:', round(precision_score(y_test, y_pred), 3))
    print('Recall score:', round(recall_score(y_test, y_pred), 3))
    print('ROC AUC score:', round(roc_auc_score(y_test, estimator.predict_proba
```

Baseline Model (Dummy Model)

In [9]:
```python
# instantiate DummyClassifier
dum = DummyClassifier(strategy='most_frequent')

# fit training data
dum.fit(X_train, y_train)
```

Out[9]:
```
DummyClassifier(strategy='most_frequent')
```

In [10]:
```python
# creating Dummy Pipeline
dum_pipe = Pipeline([
    ('ss', StandardScaler()),
    ('dummy_model', DummyClassifier(strategy='most_frequent'))
])
```

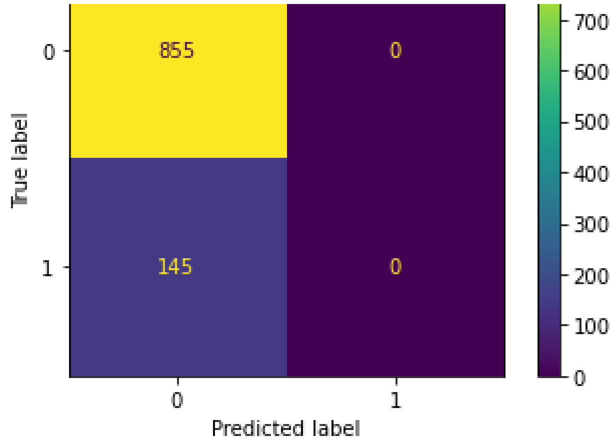
In [11]:
```python
dum_pipe.fit(X_train, y_train)
```

Out[11]:
```
Pipeline(steps=[('ss', StandardScaler()),
                ('dummy_model', DummyClassifier(strategy='most_frequent'))])
```

In [12]:
```python
print_metrics(dum_pipe, X_test, y_test)
```

```
Accuracy score: 0.855
Precision score: 0.0
Recall score: 0.0
ROC AUC score: 0.5
```

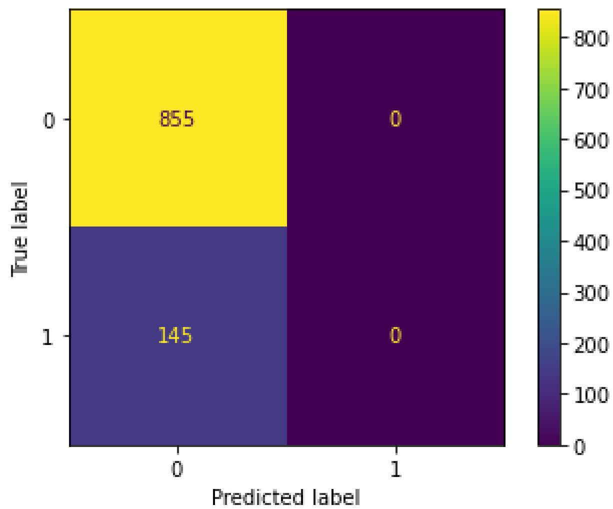
```
C:\Users\ghall\anaconda3\envs\learn-env\lib\site-packages\sklearn\metrics\_class
ification.py:1221: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control t
his behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```



In [13]:

```
plot_confusion_matrix(dum,
                      X=X_test,
                      y_true=y_test)
```

Out[13]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x15ee7bfc820>



In [14]:

```
# metrics
y_pred = dum.predict(X_test)

print('Accuracy score:', accuracy_score(y_test, y_pred))
print('Precision score:', precision_score(y_test, y_pred))
print('Recall score:', recall_score(y_test, y_pred))
```

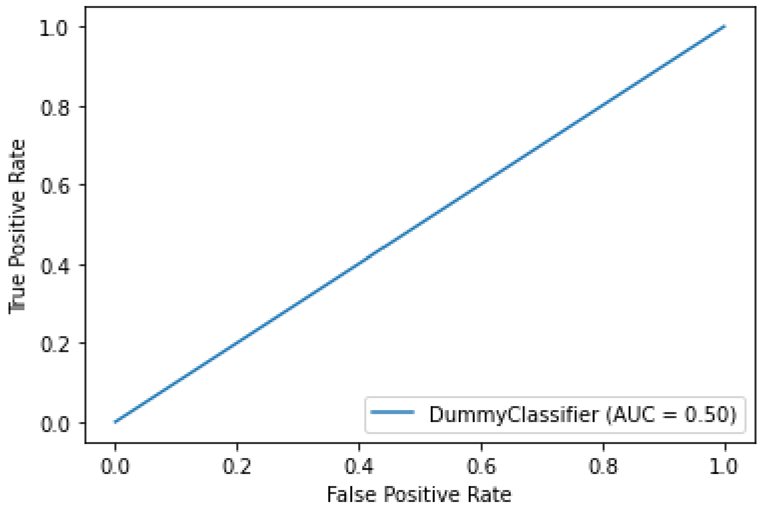
```
Accuracy score: 0.855
Precision score: 0.0
Recall score: 0.0
```

```
C:\Users\ghall\anaconda3\envs\learn-env\lib\site-packages\sklearn\metrics\_class
ification.py:1221: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control t
his behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [15]:

```
# ROC_AUC curve
plot_roc_curve(dum,
               X=X_test,
               y=y_test)
```

Out[15]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x15ee7aecd60>
```



Dummy Model:

- Dummy model is set up to only predict the most frequent class which in this case is 0 (customers who have not churned).
- Metrics for this are accuracy of 85.5%, precision of 0%, recall of 0%, and ROC_AUC of 0.50.

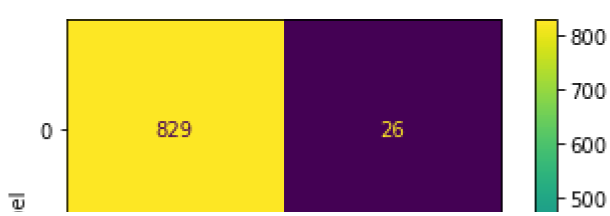
Logistic Regression

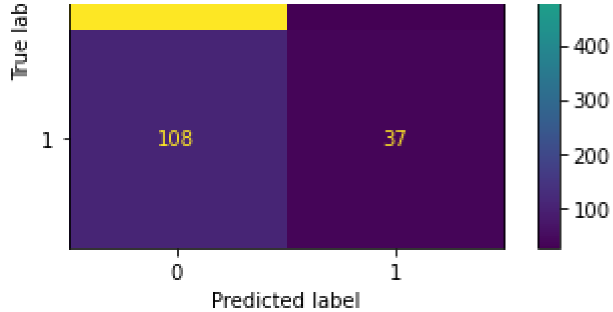
Next model to set up is a logistic regression model.

In [16]:
```
# logreg pipeline
logreg_pipe = Pipeline([
    ('ss', StandardScaler()),
    ('logreg', LogisticRegression(penalty='none'))
])
```

In [17]:
```
# fit and get metrics for logistic regression
logreg_pipe.fit(X_train, y_train)
logreg_pipe.score(X_test, y_test)

# print metrics
print_metrics(logreg_pipe, X_test, y_test)
```

```
Accuracy score: 0.866
Precision score: 0.587
Recall score: 0.255
ROC AUC score: 0.813
```





Now I am going to try to use the features that are most correlated to churn.

In [18]:
```
# top 3 correlated features
high_corr_feats = df2.corr().churn.sort_values(ascending=False).index[1:4]
print(high_corr_feats)
```

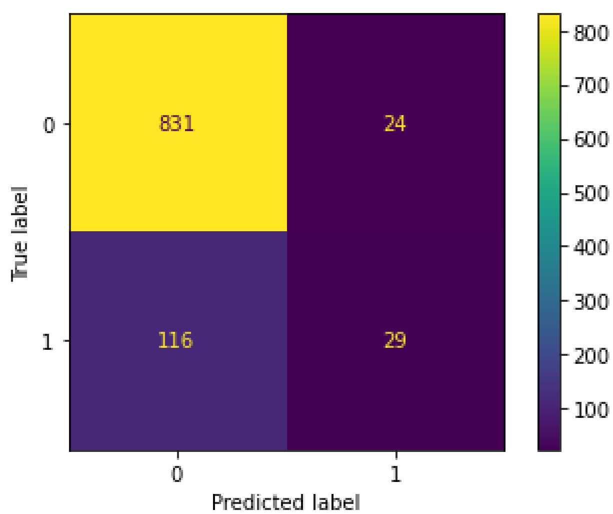
```
Index(['international plan', 'customer service calls', 'total day minutes'], dty
pe='object')
```

In [19]:
```
# fit it to logistic regression model
logreg_pipe.fit(X_train[high_corr_feats], y_train)
```

Out[19]:
```
Pipeline(steps=[('ss', StandardScaler()),
                ('logreg', LogisticRegression(penalty='none'))])
```

In [20]:
```
# print metrics
print_metrics(logreg_pipe, X_test[high_corr_feats], y_test)
```

```
Accuracy score: 0.86
Precision score: 0.547
Recall score: 0.2
ROC AUC score: 0.809
```



Logistic Regression did not perform as well as I'd hoped. Now I will experiement with the following models

- Decision Trees
- K Nearest Neighbors

- Random Forests
- Gradient Boost Classifier
- AdaBoost Classifier
- XGBoost

I will take the best performing model out of these and run a grid search to find the most optimal model by tuning the hyperparameters.

After the grid search is complete I will evaluate all of the models and choose the best one as the final model.

Decision Tree

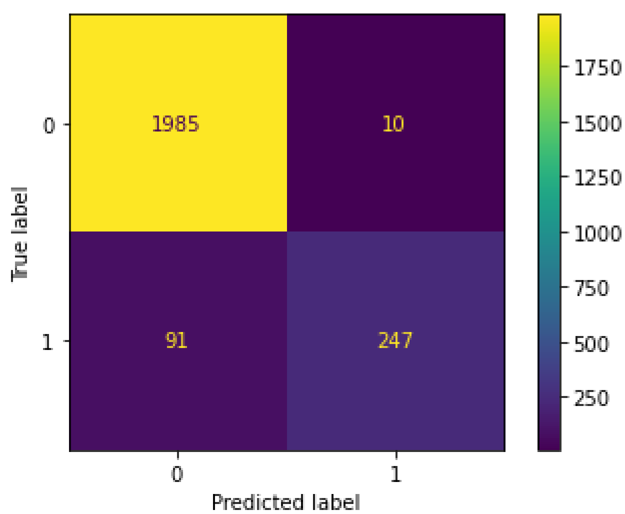
In [21]:

```
# DecisionTree pipeline
dcf_pipe = Pipeline([
    ('ss', StandardScaler()),
    ('dcf', DecisionTreeClassifier(max_depth=5))
])
```

In [22]:

```
# fit decision tree
dcf_pipe.fit(X_train, y_train)

# print train metrics
print_metrics(dcf_pipe, X_train, y_train)
```

```
Accuracy score: 0.957
Precision score: 0.961
Recall score: 0.731
ROC AUC score: 0.923
```



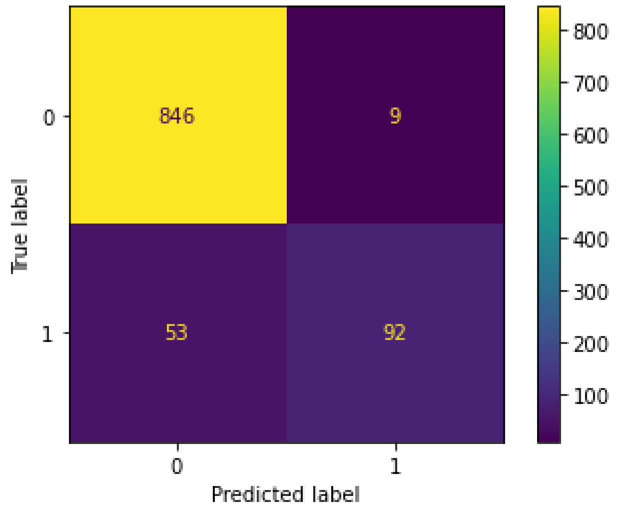
In [23]:

```
cross_val_score(dcf_pipe,
                X=X_train,
                y=y_train).mean()
```

Out[23]: 0.936996259569345

In [24]:
```python
print_metrics(dcf_pipe, X_test, y_test)
```

```
Accuracy score: 0.938
Precision score: 0.911
Recall score: 0.634
ROC AUC score: 0.822
```



K Nearest Neighbors

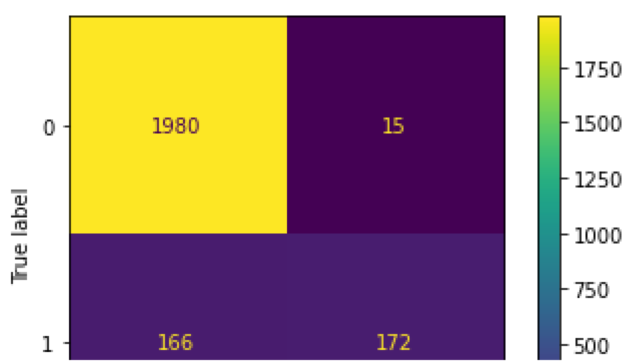
In [25]:
```python
# KNN pipeline
knn = Pipeline([
    ('ss', StandardScaler()),
    ('knn', KNeighborsClassifier())
])
```

In [26]:
```python
# fit to training data
knn.fit(X_train, y_train)
```

Out[26]:
```
Pipeline(steps=[('ss', StandardScaler()), ('knn', KNeighborsClassifier())])
```

In [27]:
```python
# print training metrics
print_metrics(knn, X_train, y_train)
```

```
Accuracy score: 0.922
Precision score: 0.92
Recall score: 0.509
ROC AUC score: 0.967
```

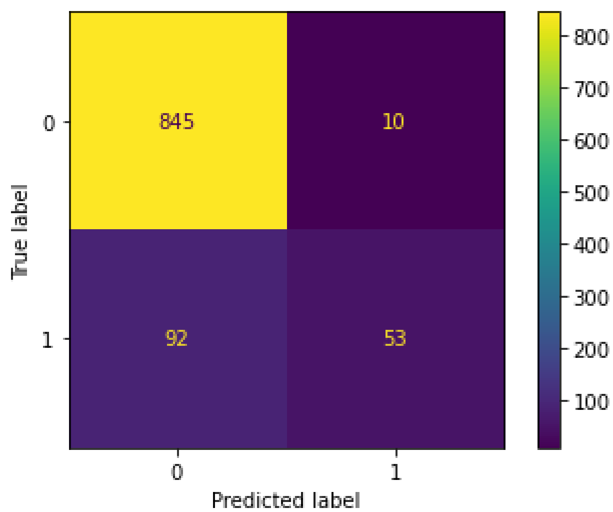


In [28]:
```
cross_val_score(knn,
               X=X_train,
               y=y_train).mean()
```

Out[28]: 0.8855575263530341

In [29]:
```
# print test metrics
print_metrics(knn, X_test, y_test)
```

```
Accuracy score: 0.898
Precision score: 0.841
Recall score: 0.366
ROC AUC score: 0.823
```

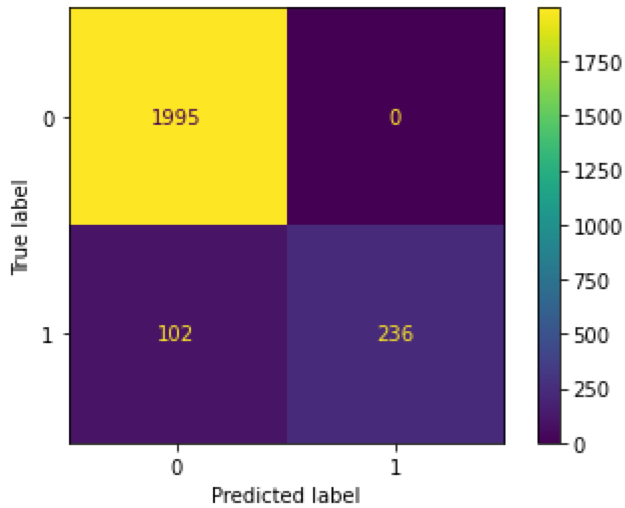


Random Forest

In [30]:
```
# Random Forest Pipeline
rcf_pipe = Pipeline([
    ('ss', StandardScaler()),
    ('rcf', RandomForestClassifier(max_depth=6))
])
```

In [31]:
```
# fit model to training data
rcf_pipe.fit(X_train, y_train)

# print training metrics
print_metrics(rcf_pipe, X_train, y_train)
```

```
Accuracy score: 0.956
Precision score: 1.0
Recall score: 0.698
ROC AUC score: 0.962
```

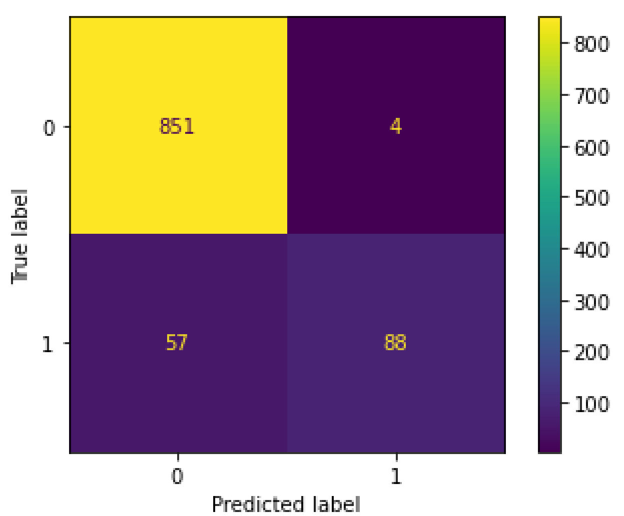


In [32]:
```
# cross validation
cross_val_score(rcf_pipe,
                X=X_train,
                y=y_train).mean()
```

Out[32]: 0.9301329828785694

In [33]:
```
print_metrics(rcf_pipe, X_test, y_test)
```

```
Accuracy score: 0.939
Precision score: 0.957
Recall score: 0.607
ROC AUC score: 0.903
```



Gradient Boost

In [34]:
```
# Gradient Boost Pipeline
gdb_pipe = Pipeline([
    ('ss', StandardScaler()),
    ('gdb', GradientBoostingClassifier(n_estimators=100))
])
```

In [35]:

```
# fit Gradient Boost
gdb_pipe.fit(X_train, y_train)
```

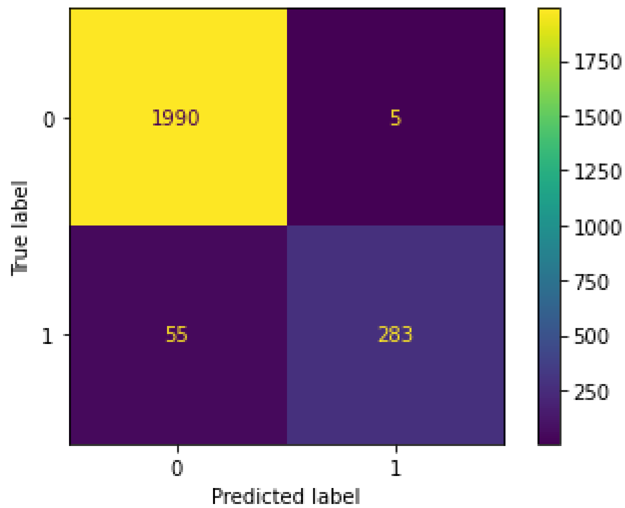
Out[35]:

```
Pipeline(steps=[('ss', StandardScaler()),
                ('gdb', GradientBoostingClassifier())])
```

In [36]:

```
# print training metrics
print_metrics(gdb_pipe, X_train, y_train)
```

```
Accuracy score: 0.974
Precision score: 0.983
Recall score: 0.837
ROC AUC score: 0.971
```



In [37]:

```
# cross validation
cross_val_score(gdb_pipe,
                X=X_train,
                y=y_train,
                scoring='recall').mean()
```

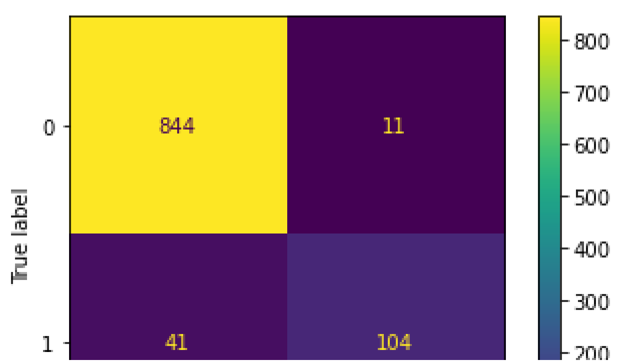
Out[37]:

```
0.727875329236172
```

In [38]:

```
# print test metrics
print_metrics(gdb_pipe, X_test, y_test)
```

```
Accuracy score: 0.948
Precision score: 0.904
Recall score: 0.717
ROC AUC score: 0.902
```



AdaBoost Classifier

In [39]:

```
# AdaBoost Classifier Pipeline
adb_pipe = Pipeline([
    ('ss', StandardScaler()),
    ('adb', AdaBoostClassifier())
])
```

In [40]:

```
# fit model to training data
adb_pipe.fit(X_train, y_train)
```

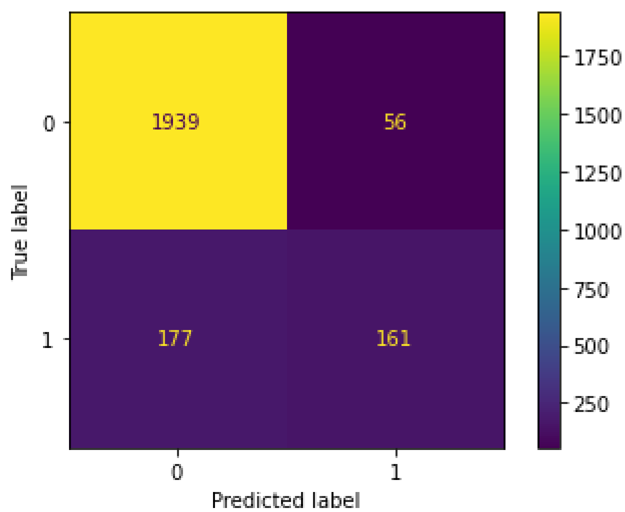
Out[40]:

```
Pipeline(steps=[('ss', StandardScaler()), ('adb', AdaBoostClassifier())])
```

In [41]:

```
# print training metrics
print_metrics(adb_pipe, X_train, y_train)
```

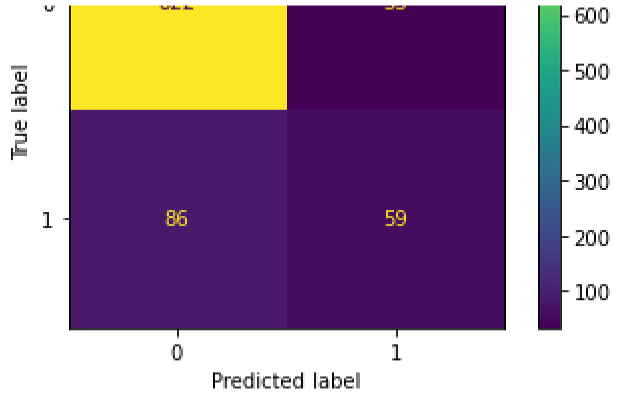
```
Accuracy score: 0.9
Precision score: 0.742
Recall score: 0.476
ROC AUC score: 0.92
```



In [42]:

```
# print training metrics
print_metrics(adb_pipe, X_test, y_test)
```

```
Accuracy score: 0.881
Precision score: 0.641
Recall score: 0.407
ROC AUC score: 0.839
```



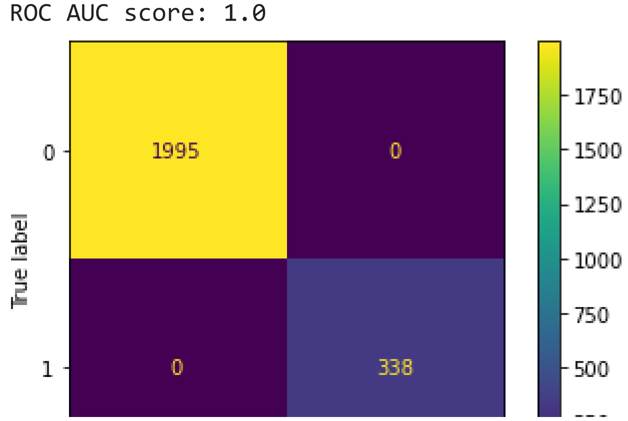
XGBoost

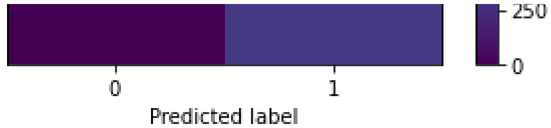
In [43]:
```
# instanstiate XGBoost
xgb = Pipeline([
    ('ss', StandardScaler()),
    ('xgb', xgboost.XGBClassifier())
])
# fit to training data
xgb.fit(X_train, y_train)
```

Out[43]:
```
Pipeline(steps=[('ss', StandardScaler()),
                ('xgb',
                 XGBClassifier(base_score=0.5, booster='gbtree',
                               colsample_bylevel=1, colsample_bynode=1,
                               colsample_bytree=1, gamma=0, gpu_id=-1,
                               importance_type='gain',
                               interaction_constraints='',
                               learning_rate=0.300000012, max_delta_step=0,
                               max_depth=6, min_child_weight=1, missing=nan,
                               monotone_constraints='()', n_estimators=100,
                               n_jobs=0, num_parallel_tree=1, random_state=0,
                               reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
                               subsample=1, tree_method='exact',
                               validate_parameters=1, verbosity=None))])
```

In [44]:
```
# print training metrics
print_metrics(xgb, X_train, y_train)
```

```
Accuracy score: 1.0
Precision score: 1.0
Recall score: 1.0
ROC AUC score: 1.0
```





In [45]:

```
# cross validation
cross_val_score(xgb,
               X=X_train,
               y=y_train).mean()
```

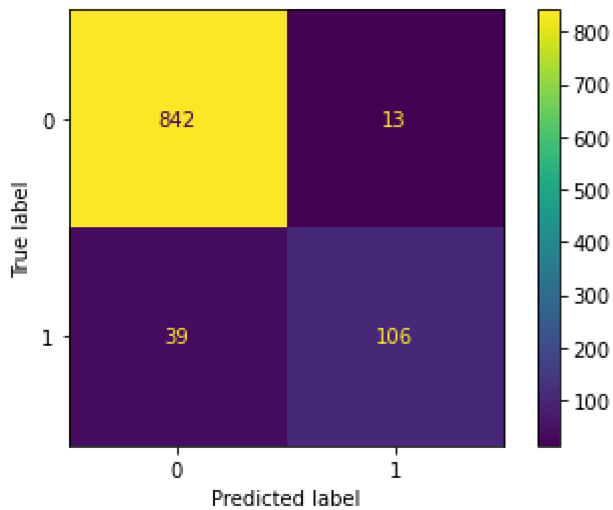
Out[45]:

```
0.9558491329001664
```

In [46]:

```
# print test metrics
print_metrics(xgb, X_test, y_test)
```

```
Accuracy score: 0.948
Precision score: 0.891
Recall score: 0.731
ROC AUC score: 0.897
```



Grid Searching on Gradient Boost

In [47]:

```
# setting up gradient boost for grid search
# no need to standardize data
gdb_gs = GradientBoostingClassifier()
```

In [48]:

```
# set paramaters for grid search
parameters = {}
parameters = {
    "max_depth":[3,5,7,9,11],
    'learning_rate': [0.01, 0.025, 0.05, 0.075, 0.1, 0.15, 0.2],
    'max_features':['log2','sqrt'],
    'criterion': ['friedman_mse', 'mae', 'mse'],
    'n_estimators':[10,25,50,75]
}
```

In [49]:

```
# scoring
```

```
# scoring
scoring = {'accuracy': make_scorer(accuracy_score),
           'recall':make_scorer(recall_score)}
```

In [50]:

```
# grid search
gs = GridSearchCV(gdb_gs, parameters, scoring=scoring,cv=5, refit='recall', ver
```

In [51]:

```
# run grid search
gs.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 840 candidates, totalling 4200 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:    2.5s
[Parallel(n_jobs=-1)]: Done 176 tasks      | elapsed:    7.0s
[Parallel(n_jobs=-1)]: Done 426 tasks      | elapsed:   16.1s
[Parallel(n_jobs=-1)]: Done 776 tasks      | elapsed:   28.4s
[Parallel(n_jobs=-1)]: Done 1226 tasks      | elapsed:   44.7s
[Parallel(n_jobs=-1)]: Done 1776 tasks      | elapsed:  3.4min
[Parallel(n_jobs=-1)]: Done 2426 tasks      | elapsed:  7.8min
[Parallel(n_jobs=-1)]: Done 3176 tasks      | elapsed: 10.6min
[Parallel(n_jobs=-1)]: Done 4026 tasks      | elapsed: 11.2min
[Parallel(n_jobs=-1)]: Done 4200 out of 4200 | elapsed: 11.4min finished
```

Out[51]:

```
GridSearchCV(cv=5, estimator=GradientBoostingClassifier(), n_jobs=-1,
             param_grid={'criterion': ['friedman_mse', 'mae', 'mse'],
                         'learning_rate': [0.01, 0.025, 0.05, 0.075, 0.1, 0.15,
                                           0.2],
                         'max_depth': [3, 5, 7, 9, 11],
                         'max_features': ['log2', 'sqrt'],
                         'n_estimators': [10, 25, 50, 75]},
             refit='recall',
             scoring={'accuracy': make_scorer(accuracy_score),
                      'recall': make_scorer(recall_score)},
             verbose=1)
```

In [52]:

```
gs.best_estimator_
```

Out[52]:

```
GradientBoostingClassifier(learning_rate=0.15, max_depth=5, max_features='sqrt',
                           n_estimators=75)
```

In [53]:

```
gs.best_score_
```

Out[53]:

```
0.7308604038630377
```

In [54]:

```
gs.cv_results_['mean_test_recall'].max()
```

Out[54]:

```
0.7308604038630377
```

In [55]:

```
pd.set_option('display.max_columns', None)
pd.DataFrame.from_dict(gs.cv_results_).sort_values('mean_test_recall', ascendin
```

Out[55]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_lear
215	0.314958	0.027992	0.006582	1.738915e-03	friedman_mse	

215	0.314958	0.027992	0.006382	1.738915e-03	friedman_mse
739	0.681180	0.017400	0.007579	4.884803e-04	mse
827	1.300524	0.063105	0.011170	1.595843e-03	mse
735	0.338296	0.009386	0.006981	3.814697e-07	mse
787	1.377119	0.057471	0.011170	3.645207e-03	mse
...	...	...	...	...	...
329	2.661286	0.120494	0.005984	2.132481e-07	mae
328	1.051589	0.091517	0.005186	3.989459e-04	mae
325	1.719604	0.158800	0.005984	6.309020e-04	mae
324	0.652655	0.084329	0.005984	6.308265e-04	mae
0	0.025732	0.002631	0.005585	1.352649e-03	friedman_mse

840 rows × 26 columns

In [56]:

```
pd.DataFrame.from_dict(gs.cv_results_).sort_values('mean_test_recall', ascendin
```

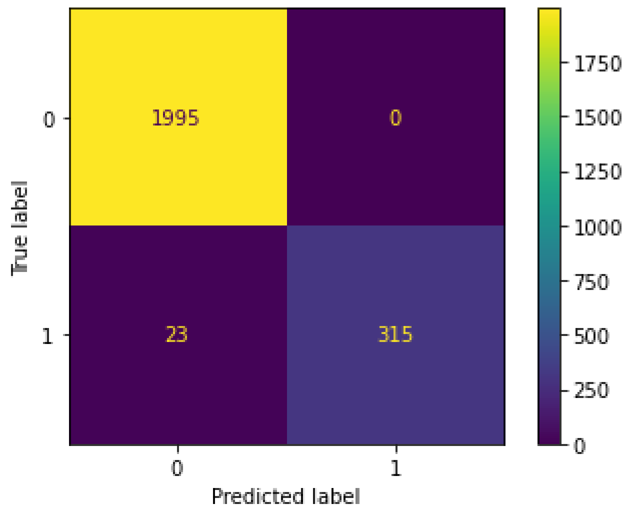
Out[56]:

```
{'criterion': 'friedman_mse',
 'learning_rate': 0.15,
 'max_depth': 5,
 'max_features': 'sqrt',
 'n_estimators': 75}
```

In [57]:

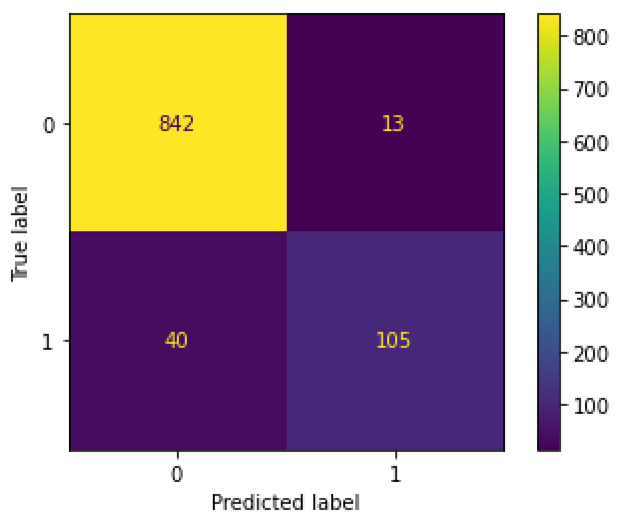
```
print_metrics(gs.best_estimator_.fit(X_train, y_train), X_train, y_train)
```

```
Accuracy score: 0.99
Precision score: 1.0
Recall score: 0.932
ROC AUC score: 0.998
```



In [58]:
```
print_metrics(gs.best_estimator_.fit(X_train, y_train), X_test, y_test)
```

```
Accuracy score: 0.947
Precision score: 0.89
Recall score: 0.724
ROC AUC score: 0.909
```



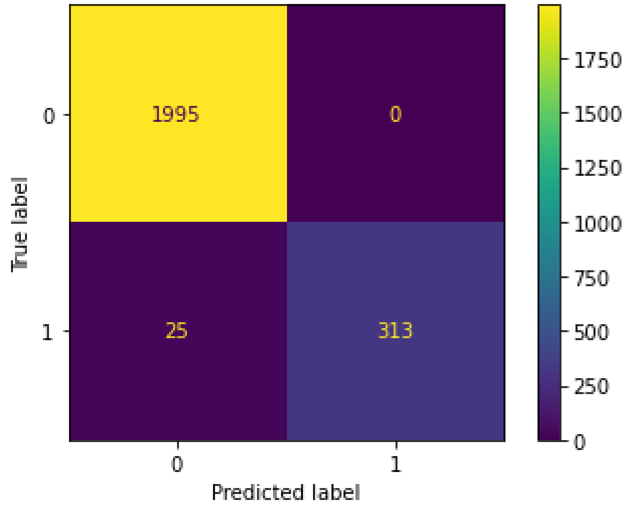
The best model from the gridsearch is overfitting a bit. To combat this I'll try out some different combinations of the hyperparameters to see if I can fix the overfitting.

In [59]:
```
gdb = GradientBoostingClassifier(learning_rate=0.2, max_depth=5, max_features='
                                 n_estimators=50, subsample=1)

gdb.fit(X_train, y_train)
```

Out[59]:
```
GradientBoostingClassifier(learning_rate=0.2, max_depth=5, max_features='sqrt',
                           n_estimators=50, subsample=1)
```

In [60]:
```
print_metrics(gdb, X_train, y_train)
```

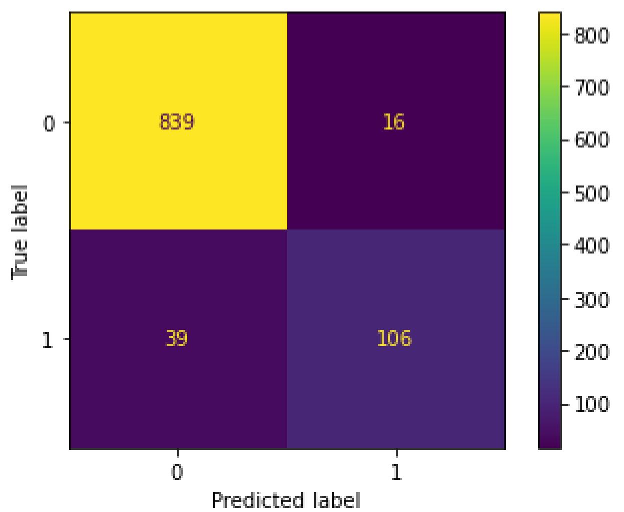
```
Accuracy score: 0.989
Precision score: 1.0
Recall score: 0.926
ROC AUC score: 0.997
```



In [61]:

```
print_metrics(gdb, X_test, y_test)
```

```
Accuracy score: 0.945
Precision score: 0.869
Recall score: 0.731
ROC AUC score: 0.911
```



I was not successful in making it less overfit.

Final Model

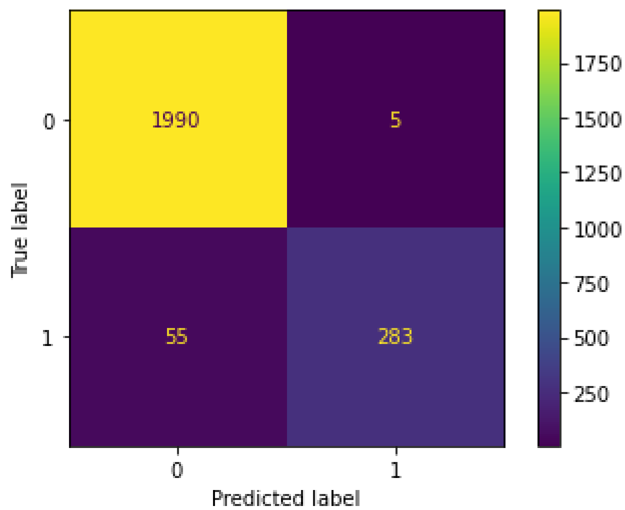
Out of all the models I trained, the best model I am going to go with is the original gradient boost classifier. This model performed well enough in terms of accuracy, recall, and ROC AUC. It is slightly overfit but less so than the other models that performed as well on the test data.

In [62]:

```
# running the gdb_pipe one more time
```

```python
gdb_pipe.fit(X_train, y_train)

print_metrics(gdb_pipe, X_train, y_train)
```

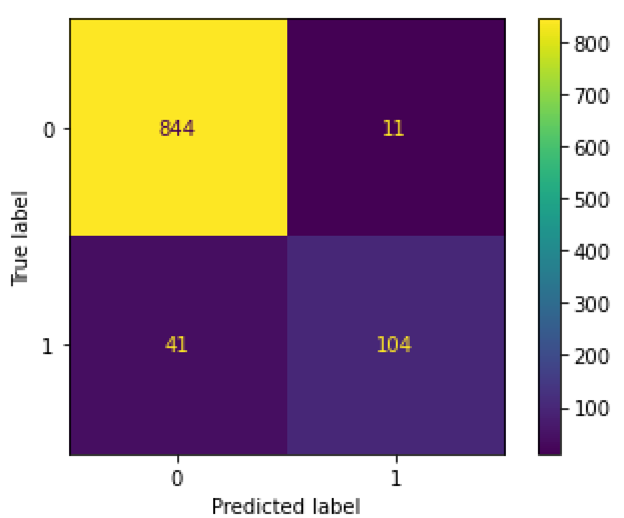
```
Accuracy score: 0.974
Precision score: 0.983
Recall score: 0.837
ROC AUC score: 0.971
```



In [63]:

```python
# print test metrics
print_metrics(gdb_pipe, X_test, y_test)
```

```
Accuracy score: 0.948
Precision score: 0.904
Recall score: 0.717
ROC AUC score: 0.901
```



In [64]:

```python
# list of feature importances
feat_imp = gdb_pipe['gdb'].feature_importances_
```

In [65]:

```python
# list of features
features = X_train.columns
```

In [66]:

```python
# creating dataframe with the features and their weights
```

```
# creating dataframe with the features and their weights
feature_df = pd.DataFrame(list(zip(features, feat_imp)),
                          columns=['features', 'feature_importance'])

feature_df = feature_df.sort_values('feature_importance', ascending=False)
```

In [67]:

```
# feature importance bar chart
sns.barplot(orient='h', data=feature_df.iloc[0:5], y='features', x='feature_imp
```

Out[67]: <AxesSubplot:xlabel='feature_importance', ylabel='features'>