Overview:

This data contains information about a direct marketing campaigns of Portuguese Bank (source: Bank Marketing (https://archive.ics.uci.edu/dataset/222/bank+marketing)). This notebook aims to analyze factors that affect the direct marketing campaigns and build a classification model to identify potential clients that would subscribe to the term deposit.

Result:

- A **stacking model** is used with 3 base models: logistic regression, random forest, gradient boosting.
- After proper cross validation and hyperparameter tuning, model achineves an **AUC score = 0.8**.
- The highest achievable recall is 0.78. We can vary threshold to attain the best combination to the specific use case.

Additionally, I also explored how each model used features provied and ranked them based on the importances.

- **Social and economic context seems to be more important than personal attributes.** In all three models, nr.employed is recognized as the most important features followed by poutcome and euribor3m.
- Month is also an important indicator as people are more likely to subscribe during some months over others. However, no apparent seasonal or other patterns were observed.
- The most important personal attribute is age.

In the future:

I hope to try neural network or other boosting model like XGBoost or LightGBM (which can not be installed to local machine due to some errors)

In [1]:
```
# https://archive.ics.uci.edu/dataset/222/bank+marketing
# https://www.sciencedirect.com/science/article/abs/pii/S016792361400061X?via%
3Dihub
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn import preprocessing
from sklearn import linear_model, ensemble
from sklearn import metrics
import warnings
import numpy as np
```

In [2]:
```
warnings.filterwarnings('ignore')
```

In [3]:
```
# df = pd.read_csv('bank-additional.csv')
df_full = pd.read_csv('bank-additional-full.csv')
```

In [4]:
```
df_full['y'] = (df_full['y'] == 'yes').astype(int) #convert y into 0, 1
df_full.sample(5)
```

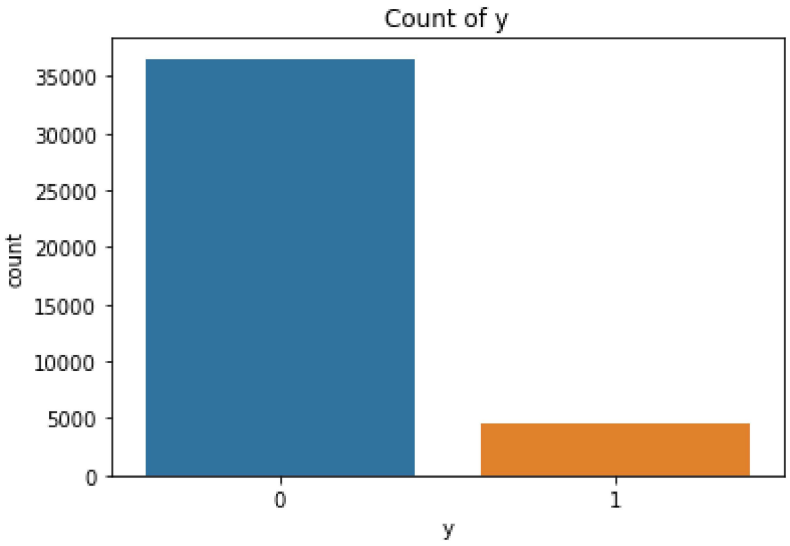
Out[4]:

	age	job	marital	education	default	housing	loan	contact	month	da
24453	54	admin.	married	high.school	no	yes	no	cellular	nov	
23088	55	admin.	married	university.degree	no	yes	no	cellular	aug	
21718	37	services	married	high.school	no	no	no	cellular	aug	
5476	49	management	married	university.degree	unknown	yes	no	telephone	may	
26728	34	student	divorced	university.degree	no	yes	no	cellular	nov	

5 rows × 21 columns

In [5]:
```
# y is highly imbalanced, I will use StratifiedKFold to Split Data and do Cros
s Validate
p = sns.countplot(x = df_full['y'])
p.set(title = 'Count of y')
```

Out[5]:
```
[Text(0.5, 1.0, 'Count of y')]
```



In [6]:
```
X_train, X_test, y_train, y_test = model_selection.train_test_split(df_full.dr
op('y', axis = 1), df_full.y,
                                                                    test_size
= 0.2, random_state = 42,
                                                                    stratify =
df_full.y)
X_train.reset_index(drop = True, inplace = True)
y_train.reset_index(drop = True, inplace = True)
X_test.reset_index(drop = True, inplace = True)
y_test.reset_index(drop = True, inplace = True)
```

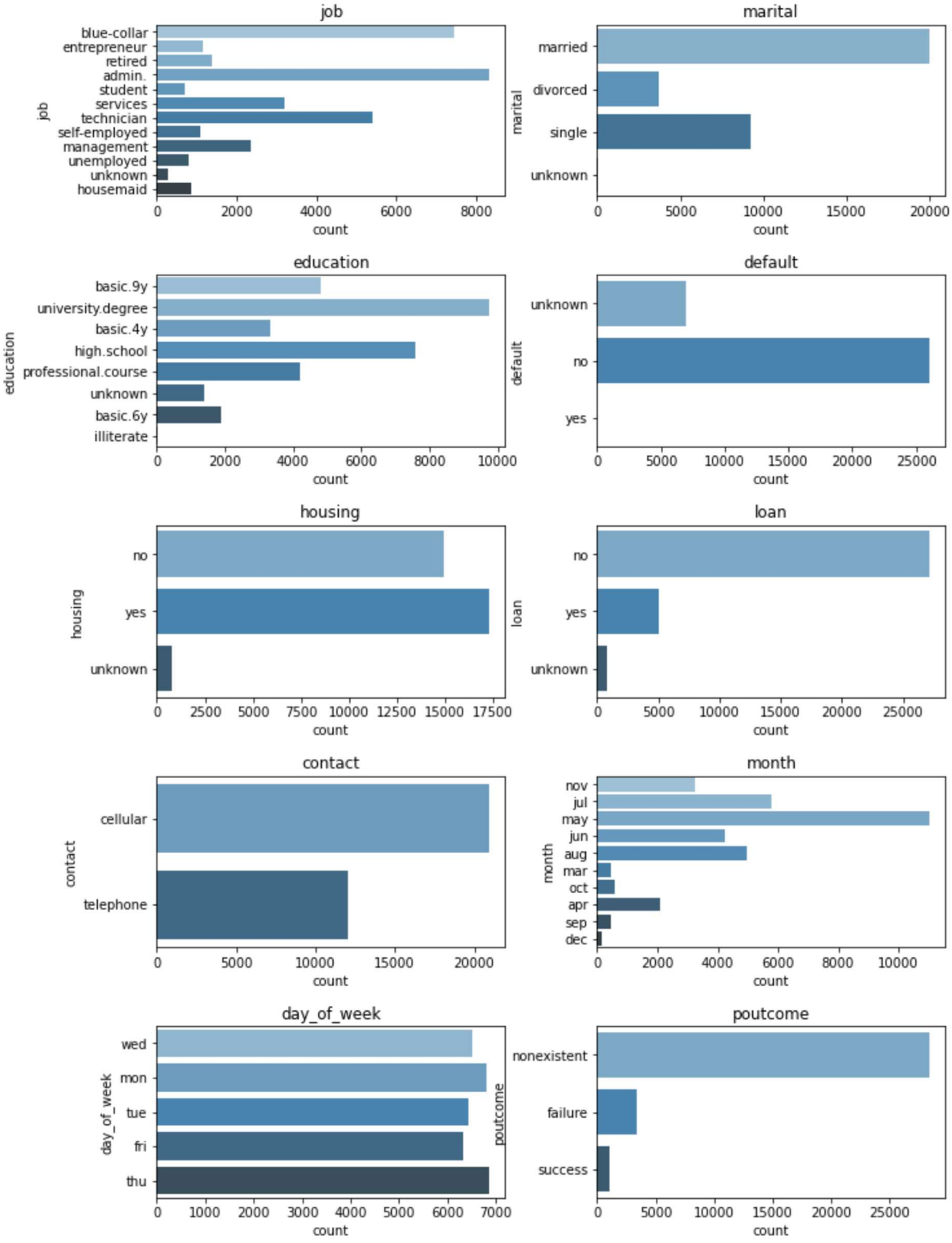
In [7]:
```
df = X_train.copy()
df['y'] = y_train
```

In [8]:
```
def make_k_folds(data, n = 5):
    """
    This function creates stratified k-folds split on given data.
    :param n: # of splits
    :returns: df with a col 'k-folds' that indicates the split
    """
    data['kfold'] = -1

    kf = model_selection.StratifiedKFold(n_splits = n, shuffle = True, random_
state = 42)

    # f: folds, t_: not sure, v_: idx of split
    for f, (t_, v_) in enumerate(kf.split(X = data, y = data['y'])):
        data.loc[v_, 'kfold'] = f

    return data
```

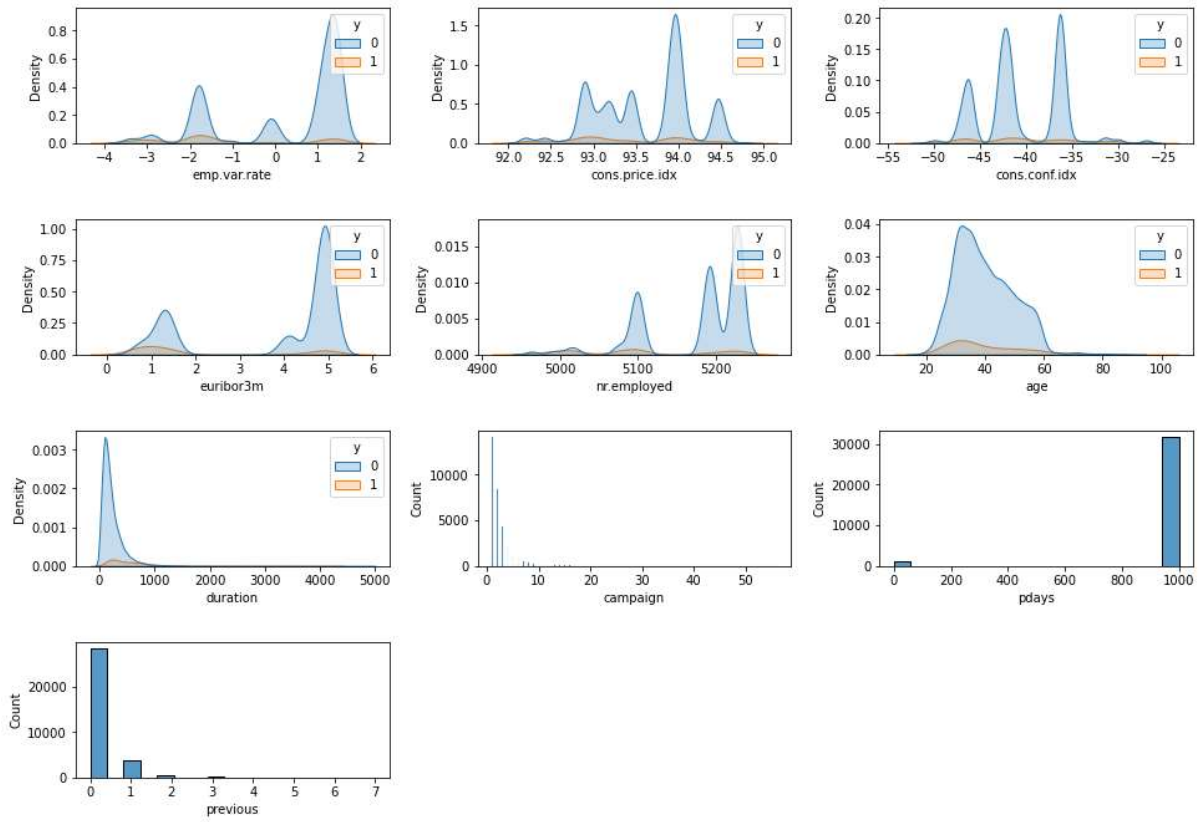
In [9]:
```
df = make_k_folds(df, n = 5)
```

Univariate Analysis

In [10]:
```
# categorical variable
cat_var = [f for f in list(df.dtypes[df.dtypes == 'object'].index) if not f == 
'y']
fig, axes = plt.subplots(5, 2, figsize = (10, 14))

fig.tight_layout(pad = 4)

for i, c in enumerate(cat_var):
    axi = axes[i//2, i%2]
    sns.countplot(y = df[c], ax = axi, palette = "Blues_d")
    axi.set(title = c)
```


job
blue-collar
entrepreneur
retired
admin.
student
services
technician
self-employed
management
unemployed
unknown
housemaid
0
2000
4000
6000
8000
count
marital
married
divorced
single
unknown
0
5000
10000
15000
20000
count
education
basic.9y
university.degree
basic.4y
high.school
professional.course
unknown
basic.6y
illiterate
0
2000
4000
6000
8000
10000
count
default
unknown
no
yes
0
5000
10000
15000
20000
25000
count
housing
no
yes
unknown
0
2500
5000
7500
10000
12500
15000
17500
count
loan
no
yes
unknown
0
5000
10000
15000
20000
25000
count
contact
cellular
telephone
0
5000
10000
15000
20000
count
month
nov
jul
may
jun
aug
mar
oct
apr
sep
dec
0
2000
4000
6000
8000
10000
count
day_of_week
wed
mon
tue
fri
thu
0
1000
2000
3000
4000
5000
6000
7000
count
poutcome
nonexistent
failure
success
0
5000
10000
15000
20000
25000
count

In [11]:
```
# continous variable
# num_var_int = list(df.dtypes[(df.dtypes == 'int64')].index)
# num_var_float = list(df.dtypes[(df.dtypes == 'float64')].index)

num_var_int = ['campaign', 'pdays', 'previous']
num_var_float = ['emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3
m', 'nr.employed', 'age', 'duration']

fig, axes = plt.subplots(4, 3, figsize = (14, 10))

fig.tight_layout(pad = 4)
fig.set_facecolor('white')

for i, c in enumerate(num_var_float):
    sns.kdeplot(data = df, x = c, hue = 'y', fill = True, ax = axes[i//3, i%
3])


for i, c in enumerate(num_var_int):
    if c == 'kfold':
        continue
    sns.histplot(data = df, x = c, ax = axes[(i+len(num_var_float))//3, (i+len
(num_var_float))%3])


plt.delaxes()
plt.delaxes(axes[3, 1])
```

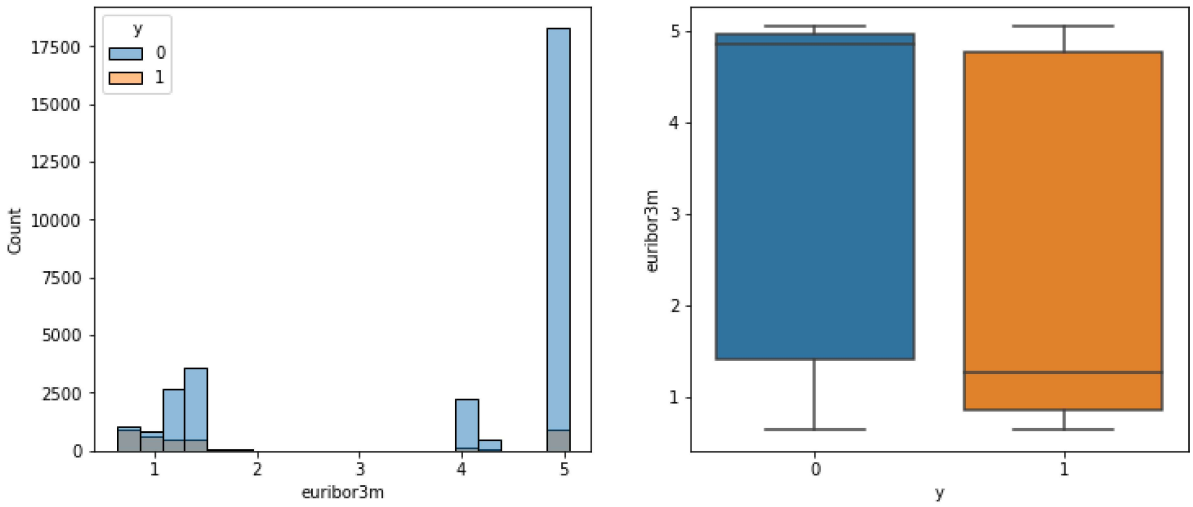


Bivariate Analysis

based on intuition and univariate analysis

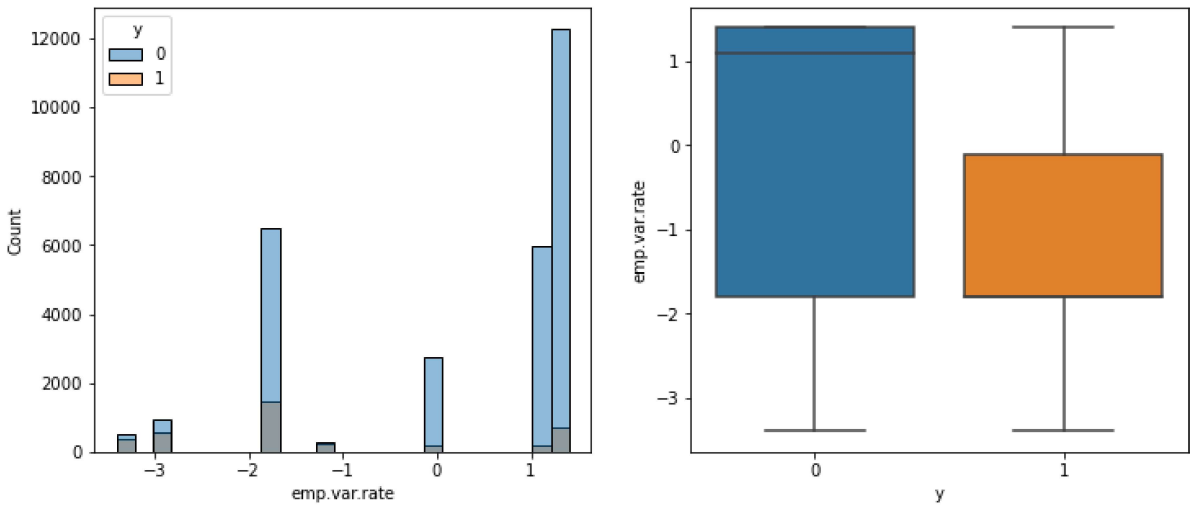
In [12]:
```
cat_var = list(df.dtypes[df.dtypes == 'object'].index)

fig, axes = plt.subplots(1, 2, figsize = (12, 5))

sns.histplot(data = df, x = 'euribor3m', hue = 'y', ax = axes[0])
sns.boxplot(data = df, x = 'y', y = 'euribor3m', ax = axes[1])
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x2c68784a0c8>



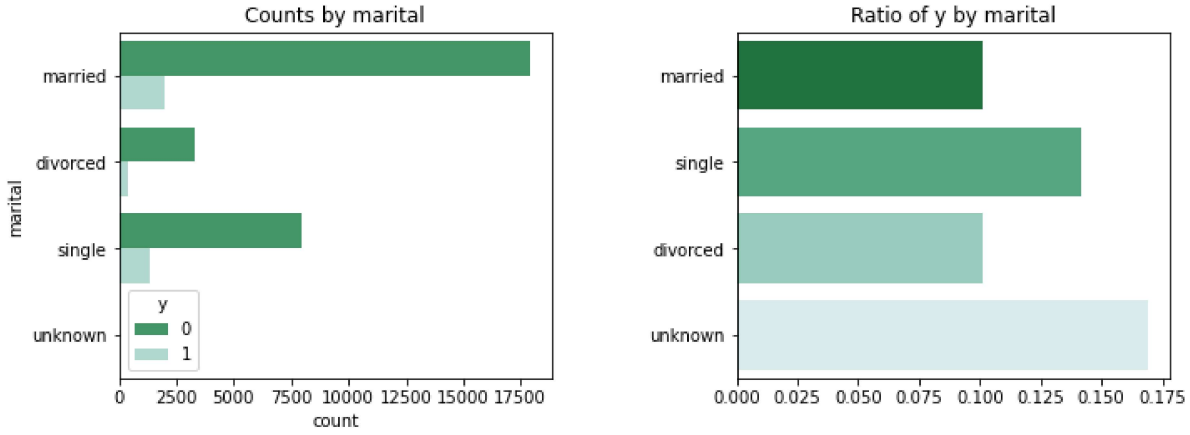
In [13]:
```
cat_var = list(df.dtypes[df.dtypes == 'object'].index)

fig, axes = plt.subplots(1, 2, figsize = (12, 5))

sns.histplot(data = df, x = 'emp.var.rate', hue = 'y', ax = axes[0])
sns.boxplot(data = df, x = 'y', y = 'emp.var.rate', ax = axes[1])
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x2c68835ff48>

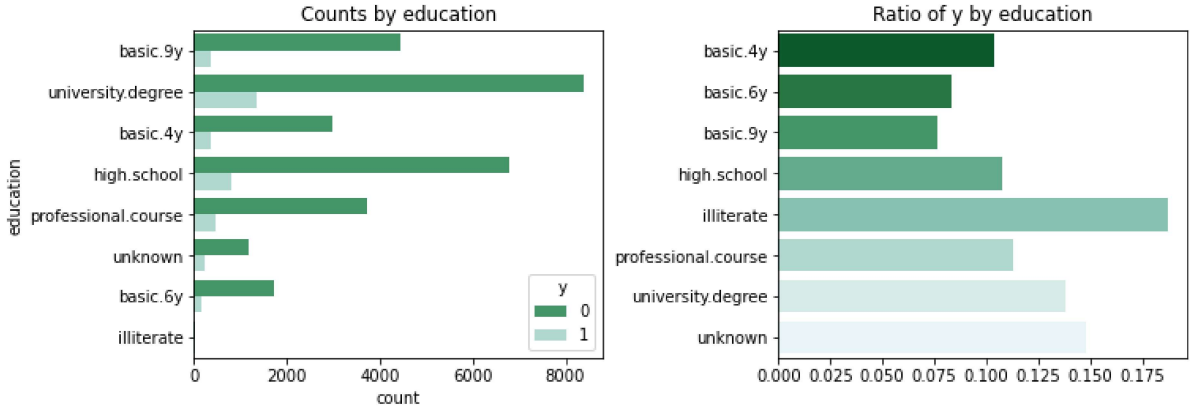


In [14]:
```
def make_cnt_ratio_plot(df, x):
    """
    This function create two plots for a given categorical x
    plot1: countplot by y
    plot2: ratio of y by catgories
    """
    fig, axes = plt.subplots(1, 2, figsize = (10,4))
    fig.tight_layout(pad = 3, w_pad = 8)

    sns.countplot(data = df, y = x, hue = 'y', ax = axes[0], palette = 'BuGn_
r')
    axes[0].set(title = f'Counts by {x}')

    df_ = df[df['y'] == 1][x].value_counts() / df[x].value_counts()
    sns.barplot(df_.values, df_.index, orient = 'h', ax = axes[1], palette =
'BuGn_r')
    axes[1].set(title = f'Ratio of y by {x}')
```

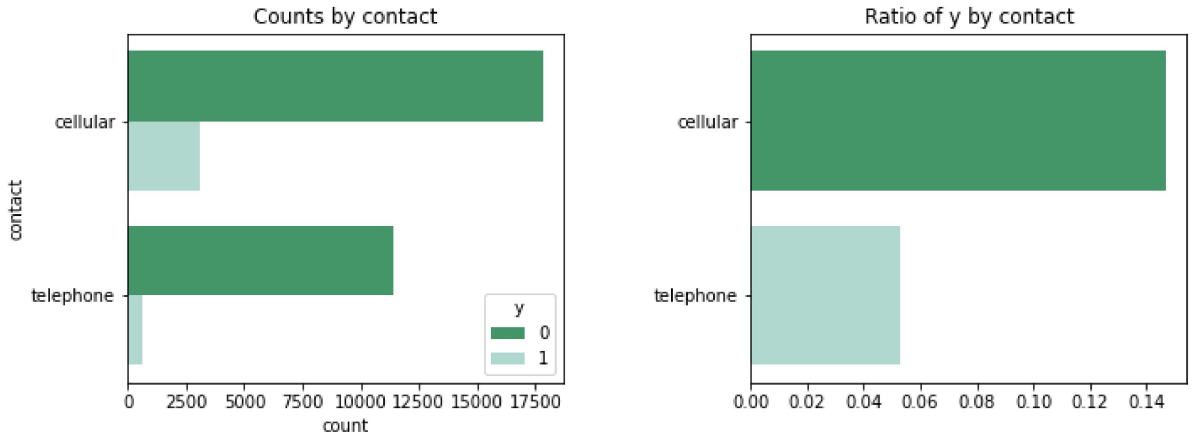
In [15]:
```
make_cnt_ratio_plot(df, 'marital')
```



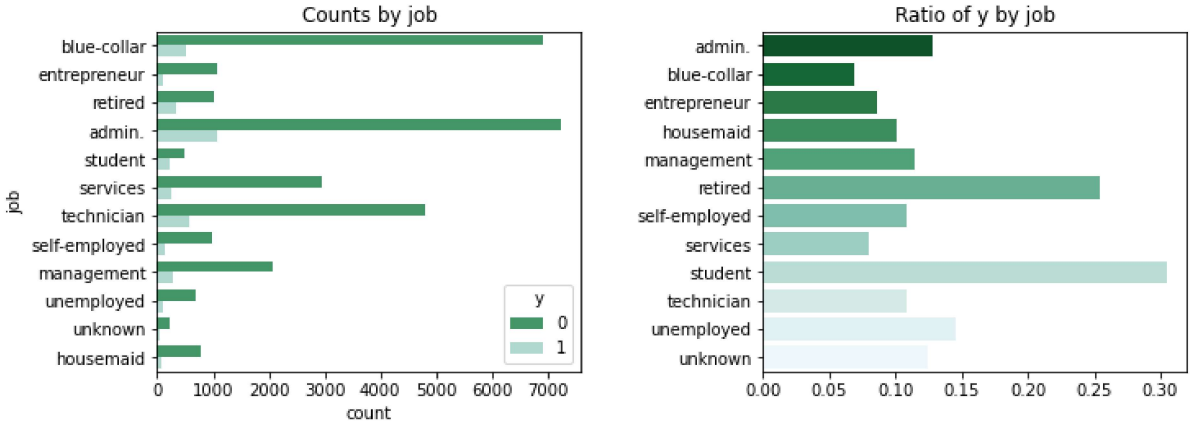
In [16]:
```
make_cnt_ratio_plot(df, 'education')
```



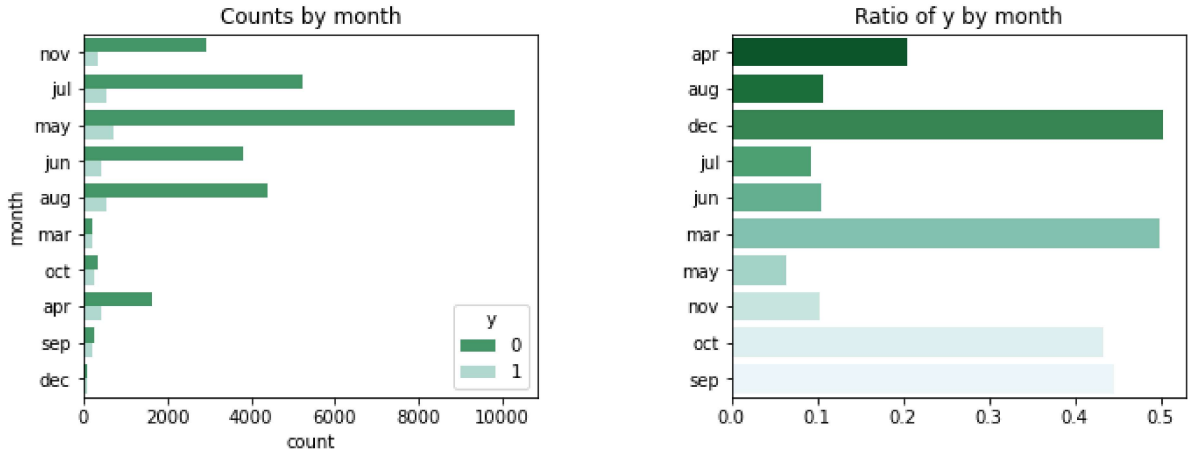
In [17]:
```
make_cnt_ratio_plot(df, 'contact')
```



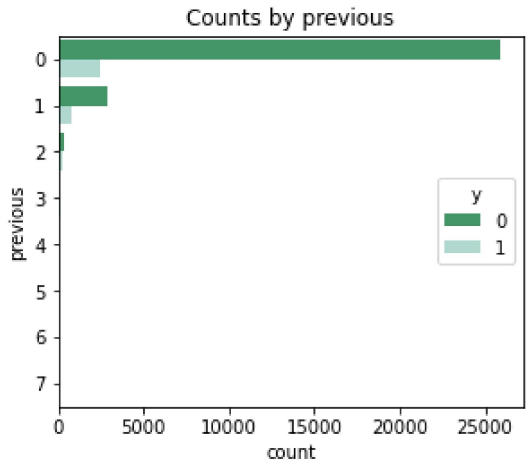
In [18]:
```
make_cnt_ratio_plot(df, 'job')
```

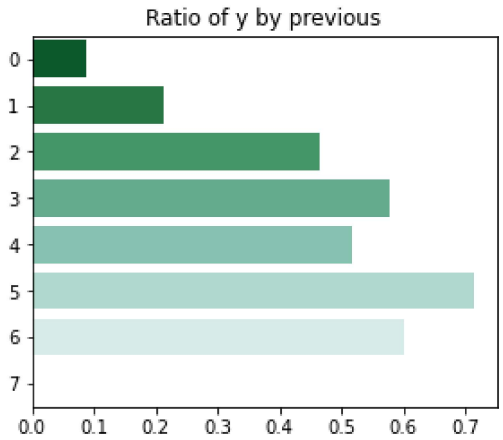


In [19]:
```
make_cnt_ratio_plot(df, 'month')
```

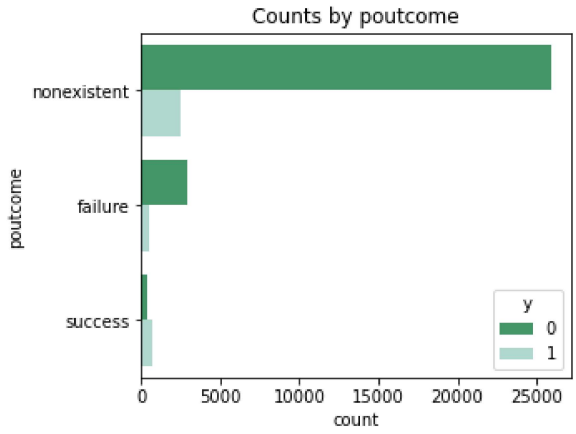


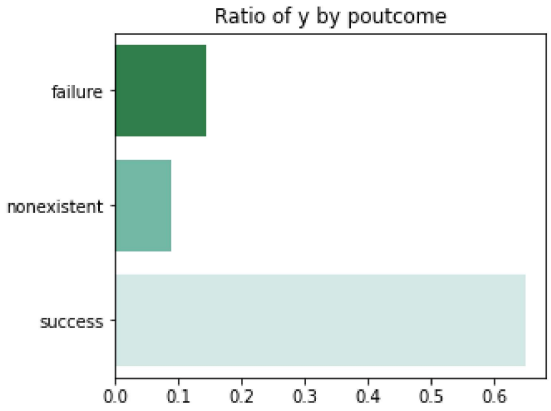
In [20]:
```
make_cnt_ratio_plot(df, 'previous')
```





In [21]:
```
make_cnt_ratio_plot(df, 'poutcome')
```





Features

- 1 - age (numeric)
- 2 - job : (cat) type of job **Consider combine**
- 3 - marital : (cat) marital status **Very few unknown, consider imputation**
- 4 - education (cat) **Very few 'illiterate', consider combine it with 'basic'**
- 5 - default: (cat) has credit in default? **very few 'yes', may not be predictive, consider drop**
- 6 - housing: (cat) has housing loan?
- 7 - loan: (cat) has personal loan?

related with the last contact of the current campaign:

- 8 - contact: (cat) contact communication type ('cellular','telephone') **0/1 convert, 1 = cellular**
- 9 - month: (cat) last contact month of year **some months has significantly less count**
- 10 - day_of_week: (cat) last contact day of the week
- 11 - duration: (numeric) last contact duration, in seconds. **Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model. Drop**

other attributes:

- 12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
- 13 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted) **~98% = 999, Consider Drop**
- 14 - previous: (numeric) number of contacts performed before this campaign and for this client **Combine Rare Cases**
- 15 - poutcome: (cat) outcome of the previous marketing campaign **majority = 'non-existent'**

social and economic context attributes:

- 16 - emp.var.rate: employment variation rate - quarterly indicator (numeric)
- 17 - cons.price.idx: consumer price index - monthly indicator (numeric)
- 18 - cons.conf.idx: consumer confidence index - monthly indicator (numeric)
- 19 - euribor3m: euribor 3 month rate - daily indicator (numeric)
- 20 - nr.employed: number of employees - quarterly indicator (numeric)

Output variable (desired target):

- 21 - y - has the client subscribed a term deposit? (binary: 'yes','no')

EDA Finding:

- People w/o job (student, unemployed, retired) are more likely to subscribe a term deposit
- People who are single are more likely to subscribe
- Generally, people with higher education levels are more likely to subscribe
- People who were contacted by cellcular are more likely to subscribe
- Poutcome is a strong predictor when we have outcomes from last campaign
- It looks like # of previous contact is positively correlated with likelihood of subscribing. But we have limited data to generalize it to previous >= 3.
- Months also seems like a important predictor of how likely a customer would subscribe term deposit. Seasonality doesn't seem to be the root of this difference. More data may needed to validate this finding.

In [22]:
```
def prep(df):

    data = df.copy()
    data = data.drop(['default', 'duration', 'pdays'], axis = 1)

    data['contact'] = (data['contact'] == 'cellular').astype(int)

    # convert education (Label Encode)
    mapping = {
        'illiterate': 0,
        'basic.4y': 0,
        'basic.6y': 0,
        'basic.9y': 0,
        'high.school': 1,
        'professional.course': 2,
        'university.degree': 3,
        'unknown': 999
    }

    data['education'] = data.education.map(mapping)

    # convert previous (used Label Encoding)
    data['previous'] = [2 if x >= 2 else x for x in data.previous]

    # Create a feature 'popular_month'
    data['popular_month'] = [1 if m in ['mar', 'dec', 'sep', 'oct'] else 0 for
m in data.month]
    data['has_job'] = [0 if j in ['unemployed', 'student', 'retired'] else 1 f
or j in data.job]

    return data
```

In [23]:
```
df_prep = prep(df)
df_prep.head(3)
```

Out[23]:

	age	job	marital	education	housing	loan	contact	month	day_of_week	campaign
0	49	blue-collar	married	0	no	no	1	nov	wed	4
1	37	entrepreneur	married	3	no	no	0	nov	wed	2
2	78	retired	married	0	no	no	1	jul	mon	1

3 rows × 21 columns

In [24]:
```
def additional_prep(df1, df2 = None):

    # extract features
    feat = [f for f in df1.columns if not f in ['y', 'kfold']]
    num_var = ['age', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euri
bor3m', 'nr.employed']

    # scale numeric variables
    scaler = preprocessing.MinMaxScaler()
    scaler.fit(df1[num_var])
    df1[num_var] = scaler.transform(df1[num_var])

    # ohe
    df1 = pd.get_dummies(df1[feat])

    if df2 is not None:
        df2[num_var] = scaler.transform(df2[num_var])
        df2 = pd.get_dummies(df2[feat])

    return df1, df2
```

In [25]:

```
def run(fold, df, model):
    """
    This function can be used for cross validation
    :param fold: int, indicate the fold we use as valid
    :param df: df used for train and valid, it has to have gone thru prep() fu
nction!
    :param model: model used to train and fit
    :print auc score
    :return fitted model
    """
    # split train & validation data
    df_train = df[df.kfold != fold].reset_index(drop = True)
    df_valid = df[df.kfold == fold].reset_index(drop = True)

    X_train, X_valid = additional_prep(df_train, df_valid)
    model.fit(X_train, df_train.y)

    valid_preds = model.predict_proba(X_valid)[:, 1]
    auc = metrics.roc_auc_score(df_valid.y, valid_preds)

    print(f'Fold {fold}, AUC score {auc.round(5)}')

    return model
```

In [26]:
```
model = linear_model.LogisticRegression()

print('--' * 10, 'logistic regression', '--' * 10)
for i in range(5):
    run(i, df_prep, model)


model = ensemble.RandomForestClassifier()

print('--' * 10, 'random forest', '--' * 10)
for i in range(5):
    run(i, df_prep, model)


model = ensemble.GradientBoostingClassifier()

print('--' * 10, 'gradient boost', '--' * 10)
for i in range(5):
    run(i, df_prep, model)
```

```
-------------------- logistic regression --------------------
Fold 0, AUC score 0.7814
Fold 1, AUC score 0.79591
Fold 2, AUC score 0.78832
Fold 3, AUC score 0.78015
Fold 4, AUC score 0.78558
-------------------- random forest --------------------
Fold 0, AUC score 0.77114
Fold 1, AUC score 0.75655
Fold 2, AUC score 0.76801
Fold 3, AUC score 0.77853
Fold 4, AUC score 0.75854
-------------------- gradient boost --------------------
Fold 0, AUC score 0.78893
Fold 1, AUC score 0.79829
Fold 2, AUC score 0.80093
Fold 3, AUC score 0.79114
Fold 4, AUC score 0.79637
```

In [27]:
```
estimators = {
    'lr': linear_model.LogisticRegression(),
    'rfc': ensemble.RandomForestClassifier(),
    'gbc': ensemble.GradientBoostingClassifier()
}

param_grid = {
    'LogisticRegression': {
        'penalty': [None, 'l1', 'l2'],
        'C': [0.01, 0.1, 1]
    },

    'RandomForestClassifier':{
        'n_estimators': [80, 120, 150],
        'max_depth': [3, 5, 8],
        'max_features': ['log2', 'sqrt'],
        'criterion': ['entropy', 'gini']
    },

    'GradientBoostingClassifier':{
        'loss': ['log_loss', 'exponential'],
        'learning_rate': [0.1, 0.3, 0.5],
        'n_estimators': [50, 100, 150]
    }
}
```

In [28]:
```
# prep df_ (entire X_train) for hyperparameter tuning
df_, _ = additional_prep(prep(df))

kf = model_selection.StratifiedKFold(n_splits = 5, shuffle = True, random_stat
e = 42)

for estimator in list(estimators.values()):
    estimator_name = estimator.__class__.__name__
    print(estimator_name)

    grid = model_selection.GridSearchCV(estimator, param_grid = param_grid[est
imator_name], cv = kf)
    grid.fit(df_, y_train)
    print(grid.best_params_)
```

```
LogisticRegression
{'C': 1, 'penalty': 'l2'}
RandomForestClassifier
{'criterion': 'entropy', 'max_depth': 8, 'max_features': 'sqrt', 'n_estimator
s': 80}
GradientBoostingClassifier
{'learning_rate': 0.1, 'loss': 'exponential', 'n_estimators': 150}
```

In [29]:
```
# after tuning
estimators_tuned = {
    'lr': linear_model.LogisticRegression(C = 1, penalty = 'l2'),

    'rfc': ensemble.RandomForestClassifier(criterion = 'entropy',
                                           max_depth = 8,
                                           max_features = 'sqrt',
                                          n_estimators = 150),

    'gbc': ensemble.GradientBoostingClassifier(learning_rate = 0.1,
                                               loss = 'exponential',
                                               n_estimators = 150)
}
```

In [30]:
```
# after tuning validation score
for e in estimators_tuned:
    model = estimators_tuned[e]
    print('--' * 10, model.__class__.__name__, '--' * 10)
    for i in range(5):
        run(i, df_prep, model)
```

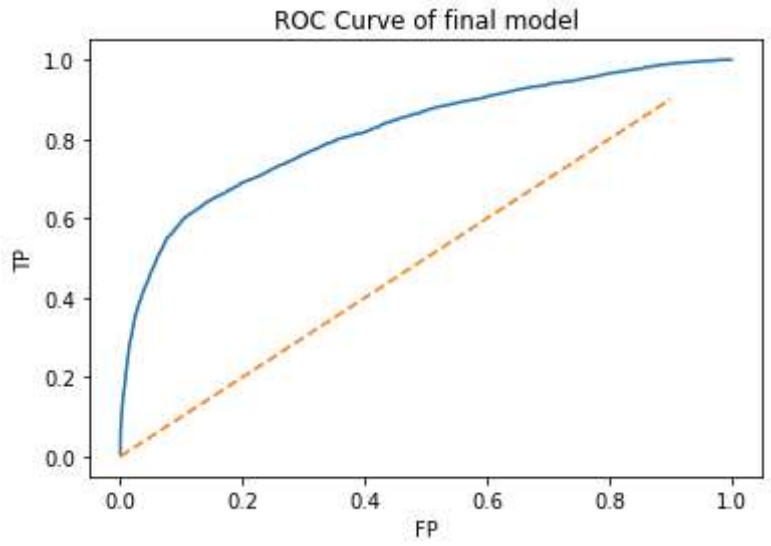
```
-------------------- LogisticRegression --------------------
Fold 0, AUC score 0.7814
Fold 1, AUC score 0.79591
Fold 2, AUC score 0.78832
Fold 3, AUC score 0.78015
Fold 4, AUC score 0.78558
-------------------- RandomForestClassifier --------------------
Fold 0, AUC score 0.79156
Fold 1, AUC score 0.7989
Fold 2, AUC score 0.79832
Fold 3, AUC score 0.79115
Fold 4, AUC score 0.79603
-------------------- GradientBoostingClassifier --------------------
Fold 0, AUC score 0.78777
Fold 1, AUC score 0.79859
Fold 2, AUC score 0.80382
Fold 3, AUC score 0.79512
Fold 4, AUC score 0.79896
```

In [31]:
```
result_train = pd.DataFrame()
result_test = pd.DataFrame()

X_train_prep, _ = additional_prep(prep(X_train))
X_test_prep, _ = additional_prep(prep(X_test))

for e in estimators_tuned:
    estimator = estimators_tuned[e]

    # fit train_X
    estimator.fit(X_train_prep, y_train)

    # predict proba of test_X
    result_train[e] = estimator.predict_proba(X_train_prep)[:, 1]
    result_test[e] = estimator.predict_proba(X_test_prep)[:, 1]
```

In [32]:
```
proba_train = result_train['lr'] * 0.1 + result_train['rfc'] * 0.2 + result_tr
ain['gbc'] * 0.7
proba_test = result_test['lr'] * 0.1 + result_test['rfc'] * 0.2 + result_test
['gbc'] * 0.7

# (fpr, tpr, threshold)
roc = metrics.roc_curve(y_true = y_train, y_score = proba_train)
```

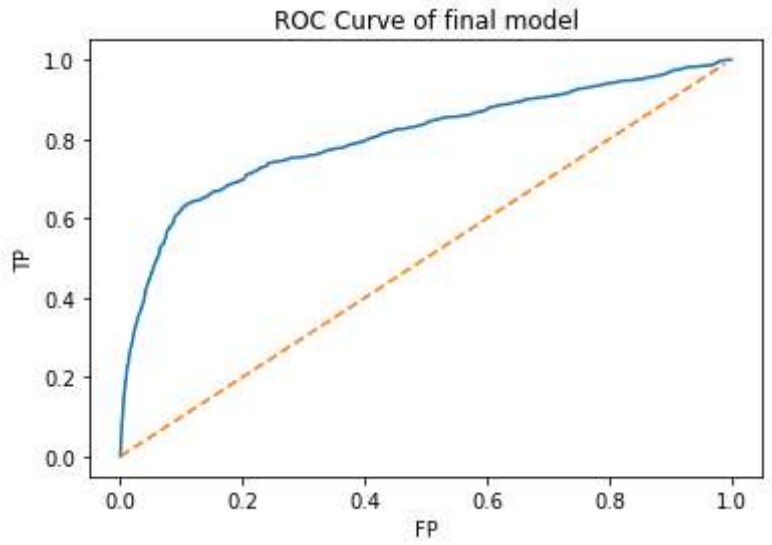
In [33]:
```
fig = plt.figure()
fig.set_facecolor('white')

ax = sns.lineplot(x = roc[0], y = roc[1])
sns.lineplot(x = np.arange(0, 1, 0.1), y = np.arange(0, 1, 0.1), linestyle='da
shed', ax = ax)
ax.set(title = 'ROC Curve of final model', xlabel = 'FP', ylabel = 'TP')
```

Out[33]:
```
[Text(0, 0.5, 'TP'),
 Text(0.5, 0, 'FP'),
 Text(0.5, 1.0, 'ROC Curve of final model')]
```



In [34]:
```
roc = metrics.roc_curve(y_true = y_test, y_score = proba_test)

fig = plt.figure()
fig.set_facecolor('white')

ax = sns.lineplot(x = roc[0], y = roc[1])
sns.lineplot(x = np.arange(0, 1, 0.01), y = np.arange(0, 1, 0.01), linestyle
='dashed', ax = ax)
ax.set(title = 'ROC Curve of final model', xlabel = 'FP', ylabel = 'TP')
```

Out[34]:
```
[Text(0, 0.5, 'TP'),
 Text(0.5, 0, 'FP'),
 Text(0.5, 1.0, 'ROC Curve of final model')]
```

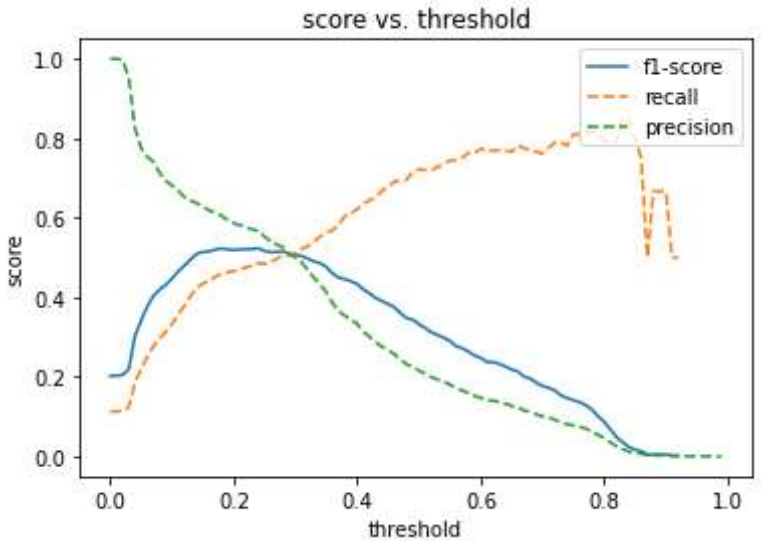


In [35]:
```
def recall(proba, y_true, threshold):
    tn, fp, fn, tp = metrics.confusion_matrix((proba > threshold).astype(int),
y_true).ravel()
    return (tp/(tp+fn)).round(4)

def precision(proba, y_true, threshold):
    tn, fp, fn, tp = metrics.confusion_matrix((proba > threshold).astype(int),
y_true).ravel()
    return (tp/(tp+fp)).round(4)

def f1_score(proba, y_true, threshold):
    r = recall(proba, y_true, threshold)
    p = precision(proba, y_true, threshold)
    return ((2*p*r)/(p+r)).round(4)
```

In [36]:
```
pr = pd.DataFrame()

for i in range(100):
    pr.loc[i, 'threshold'] = i/100
    pr.loc[i, 'recall'] = recall(proba_test, y_test, threshold = i/100)
    pr.loc[i, 'precision'] = precision(proba_test, y_test, threshold = i/100)
    pr.loc[i, 'f1'] = f1_score(proba_test, y_test, threshold = i/100)
```

In [37]:
```
fig = plt.figure()
fig.set_facecolor('white')

ax = sns.lineplot(data = pr, x = 'threshold', y = 'f1')
sns.lineplot(data = pr, x = 'threshold', y = 'recall', ax = ax, linestyle = 'd
ashed')
sns.lineplot(data = pr, x = 'threshold', y = 'precision', ax = ax, linestyle =
'dashed')
ax.set(title = 'score vs. threshold', ylabel = 'score')
plt.legend(labels = ['f1-score', 'recall', 'precision'], loc = 1)
```

Out[37]: <matplotlib.legend.Legend at 0x2c687afbf88>



In [38]:

```
fig, axes = plt.subplots(1, 3, figsize = (15, 5))

fig.tight_layout(pad = 3, w_pad = 8, h_pad = 8)
fig.suptitle('Top 10 Important Features')
fig.set_facecolor('white')

for i, key in enumerate(estimators_tuned):

    try:
        x = estimators_tuned[key].coef_[0]
    except:
        x = estimators_tuned[key].feature_importances_

    feature_name =  estimators_tuned[key].feature_names_in_
    estimator_name = estimators_tuned[key].__class__.__name__

    top_10_features = pd.DataFrame({'feature': feature_name[np.argsort(abs
(x))][-10:],
                                    'importance': x[np.argsort(abs(x))][-10:]})
    top_10_features.sort_values(by = 'importance', ascending = False, inplace
= True)

    ax = sns.barplot(data = top_10_features, x = 'importance', y = 'feature',
palette = 'BuGn', ax = axes[i])
    ax.set(title = f'{estimator_name}', ylabel = '')
```