In [1]:
```python
#Import needed packages
import pandas as pd
import numpy as np
import glob
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_selection import mutual_info_regression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,precision_score, confusion_matrix
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
```

In [2]:
```python
#Loads 201 CSV files into a dataframe.
path="C:/Users/predi/Documents/GitHub/DSC680/Assignments/Project 1/Datasets"
csv_files = glob.glob(path + "/*.csv")
raw_df = pd.concat(map(pd.read_csv, glob.glob(path + "/*.csv")))
```

In [3]:
```python
#Lower cases all headers and removes any white space from them.
raw_df.columns=raw_df.columns.str.lower()
raw_df.columns=raw_df.columns.str.strip()
#Creates a list of all column names in the dataframe
raw_df.columns
```

Out[3]:
```
Index(['id', 'timestamp', '/es', '/nq', '/rty', 'spy', 'qqq', 'iwm', 'aapl',
       'msft', 'nvda', 'xlk', 'xlf', 'xlp', 'xly', 'xtn', 'hyg', 'db_col_16',
       'db_col_17', '/es volume', 'tlt', 'tlt volume', '***', '****',
       '/es sma20', '/es sma50', '35', '2020-12-27 17:59:09.000', '3692.75',
       '12703', '2001.8', '369', '309.56', '199.01', '131.97', '222.75',
       '519.75', '129.06', '28.95', '66.72', '157.88', '71.3937', '87.05', '0',
       '0.1', '0.2', '157.29', '3117104', '145462', '2021-01-03 17:59:08.000',
       '3748.75', '12883', '1978.3', '373.88', '313.74', '196.06', '132.69',
       '222.42', '522.2', '130.02', '29.48', '67.45', '160.78', '71.37',
       '87.3', '157.73', '7742413', '290744', '2021-01-10 17:59:08.000',
       '3822', '13123.25', '2092.1', '381.26', '319.03', '207.72', '132.05',
       '219.62', '531.07', '130.76', '30.92', '66.9', '168.79', '73.3652',
       '87.37', '151.32', '1.362282e+07', 'tnx', 'tyx', 'vix', 'spx',
       'spx pcr', 'spy pcr', '/es pcr'],
      dtype='object')
```

In [4]:
```python
#Drops all columns but the three that are related to the S&P 500 future contracts.
drop_list=['id', '/nq', '/rty', 'spy', 'qqq', 'iwm', 'aapl','msft',
           'nvda', 'xlk', 'xlf', 'xlp', 'xly', 'xtn', 'hyg', 'db_col_16',
           'db_col_17','tlt', 'tlt volume', '***', '****','35',
           '2020-12-27 17:59:09.000', '3692.75','12703', '2001.8', '369',
           '309.56', '199.01', '131.97', '222.75','519.75', '129.06', '28.95',
           '66.72', '157.88', '71.3937', '87.05', '0','0.1', '0.2', '157.29',
           '3117104', '145462', '2021-01-03 17:59:08.000','3748.75', '12883',
           '1978.3', '373.88', '313.74', '196.06', '132.69','222.42', '522.2',
           '130.02', '29.48', '67.45', '160.78', '71.37','87.3', '157.73',
           '7742413', '290744', '2021-01-10 17:59:08.000','3822', '13123.25',
           '2092.1', '381.26', '319.03', '207.72', '132.05','219.62', '531.07',
           '130.76', '30.92', '66.9', '168.79', '73.3652','87.37', '151.32',
           '1.362282e+07', 'tnx', 'tyx', 'vix', 'spx','spx pcr', 'spy pcr',
           '/es sma20','/es sma50','/es pcr']
dropped_df = raw_df.drop(columns=drop_list)
#Shows striped down dataframe
dropped_df.sample(10)
```

Out[4]:

	timestamp	/es	/es volume
101342	2020-10-22 06:34:11.000	3436.50	2.0
110747	2023-04-06 14:26:26.000	4133.00	0.0
3590	2021-11-28 20:58:48.000	4637.75	82.0
123178	2023-03-24 00:48:24.000	3982.00	10.0
123993	2021-02-05 01:34:23.000	3877.50	6.0
62878	2021-04-27 22:23:09.000	4184.75	10.0
140614	2023-08-18 15:33:13.000	4381.25	155.0
83533	2021-03-10 15:47:06.000	3903.50	133.0
138768	2022-12-16 13:49:35.000	3865.50	127.0
14752	2023-02-13 06:18:07.000	4102.75	0.0

In [5]:
```
#Creates five minute candle sticks out of the 3 second observations.
dropped_df['timestamp'] = pd.to_datetime(dropped_df['timestamp'])
new_df = dropped_df.groupby(pd.Grouper(key='timestamp', freq='5T')).agg({
    'timestamp': 'last',
    '/es': ['max', 'min', 'last'],
    '/es volume': 'sum'})
# Flattens multi-level columns, resets index, and renames
new_df.columns = [f'{col[0]}_{col[1]}' for col in new_df.columns]
new_df.reset_index(drop=True, inplace=True)
new_df.rename(columns={'timestamp_last': 'last_timestamp',
                       '/es_max': 'max_es', '/es_min': 'min_es', '/es_last': 'last_es',
                       'es volume_sum': 'total_volume'}, inplace=True)
#Rounds the time stamps to the nearest minute
new_df['last_timestamp'] = pd.to_datetime(new_df['last_timestamp']).dt.round('T')
new_df.set_index('last_timestamp', inplace=True)
#Drops NaN rows created by Grouper for the weekends
new_df.dropna(axis=0,inplace=True)
```

In [6]:
```
#Creates simple moving averages columns for multiple period lengths
new_df['ma_10'] = new_df['last_es'].rolling(window=10).mean()
new_df['ma_20'] = new_df['last_es'].rolling(window=20).mean()
new_df['ma_50'] = new_df['last_es'].rolling(window=50).mean()
new_df['ma_100'] = new_df['last_es'].rolling(window=100).mean()
new_df['ma_200'] = new_df['last_es'].rolling(window=200).mean()

#Creates upper and lower Bollinger Bands columns
new_df['std'] = new_df['last_es'].rolling(window=20).std()
new_df['bb_upper'] = new_df['ma_20'] + (2 * new_df['std'])
new_df['bb_lower'] = new_df['ma_20'] - (2 * new_df['std'])
#Drops columns created for calulations
new_df.drop(columns=['std'], inplace=True)
```

In [7]:
```
#Calculate the 12-day and 26-day EMAs, to be used in MACD calulation
new_df['ema_12'] = new_df['last_es'].ewm(span=12, adjust=False).mean()
new_df['ema_26'] = new_df['last_es'].ewm(span=26, adjust=False).mean()
#Calculates the MACD
new_df['macd_rough'] = new_df['ema_12'] - new_df['ema_26']
new_df['macd'] = new_df['macd_rough'].ewm(span=9, adjust=False).mean()
#Drops columns created for calulations
new_df.drop(columns=['macd_rough'], inplace=True)
```

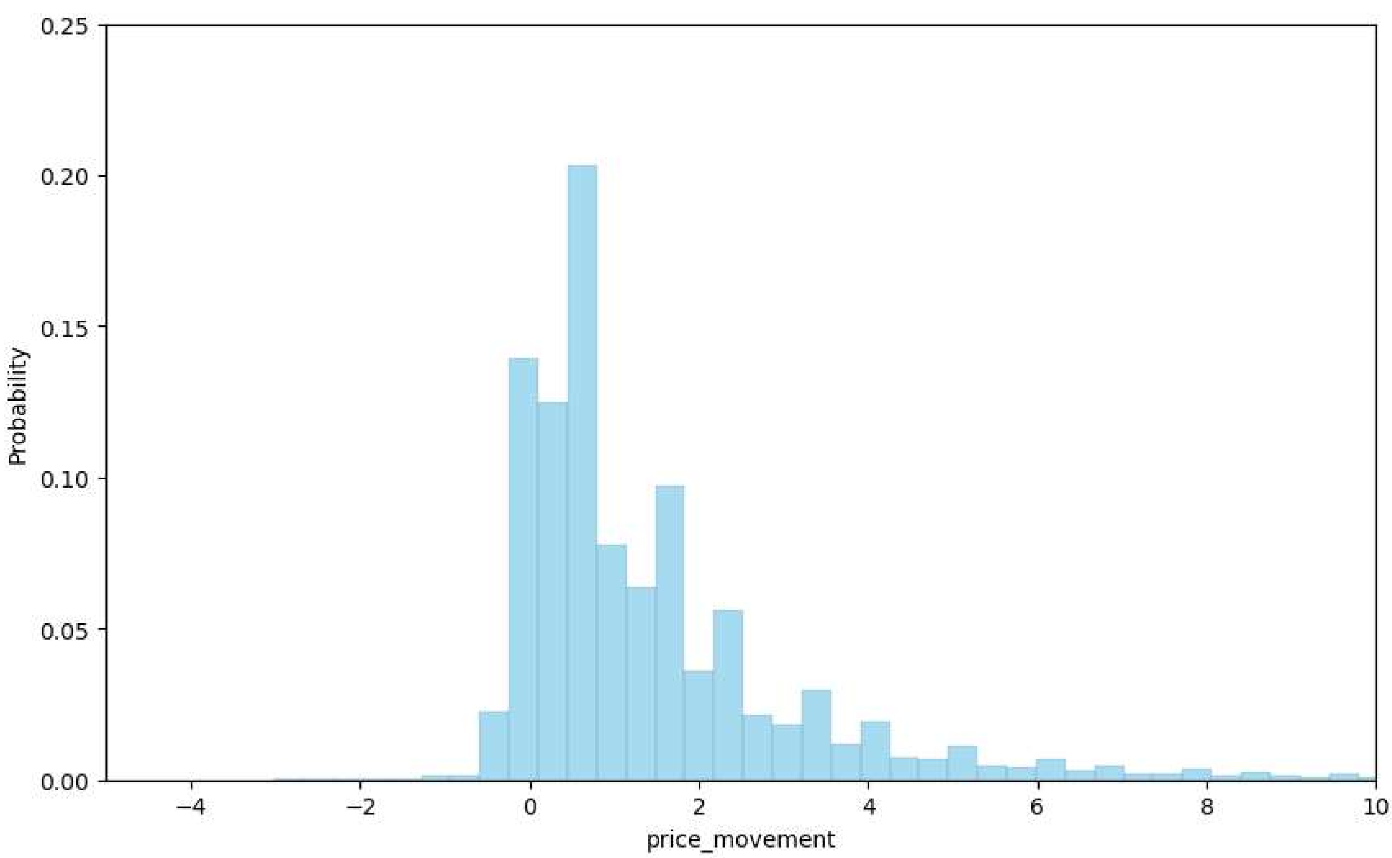
In [8]:
```
#Calculates the Stochastic Oscillator
new_df['l14'] = new_df['min_es'].rolling(window=14).min()
new_df['h14'] = new_df['max_es'].rolling(window=14).max()
new_df['k'] =((new_df['last_es'] - new_df['l14']) / (new_df['h14'] - new_df['l14'])) * 100
new_df['stochastic'] = new_df['k'].rolling(window=3).mean()
#Drops columns created for calulations
new_df.drop(columns=['l14','h14','k'], inplace=True)
```

In [9]:
```
#Calulates Fibonacci retracements over 60 minutes
retracement_levels = [0.236, 0.382, 0.500, 0.618, 0.786]
new_df['price_range'] = new_df['last_es'].rolling(window=12).max() - new_df[
    'last_es'].rolling(window=12).min()

for level in retracement_levels:
    new_df[f'fibonacci_{int(level * 100)}%'] = new_df[
        'last_es'].rolling(window=12).max() - new_df['price_range'] * level

#Drops columns created for calulations
new_df.drop(columns=['price_range'], inplace=True)
```

In [10]:
```
#Calulates VWAP on a rolling 60 minute window
new_df['tpv'] = ((new_df['min_es']+new_df['last_es']+new_df[
    'max_es'])/3)*new_df['/es volume_sum']
new_df['vwap'] = (new_df['tpv'].rolling(window=12).sum()) / (new_df[
    '/es volume_sum'].rolling(window=12).sum())

#Drops columns created for calulations
new_df.drop(columns=['tpv'], inplace=True)

#Drops rows with NaN values
new_df.dropna(inplace=True)
```

In [11]:
```
#calulates the max difference in price for the next 5 minutes
new_df['price_movement'] = new_df['max_es'].shift(-1) - new_df['last_es']
#drops the two max and min outliers created by the calculation
new_df.drop(new_df['price_movement'].idxmax(), inplace=True)
new_df.drop(new_df['price_movement'].idxmin(), inplace=True)

#Creates boolean if there was atleats a 2 point increase during the period
new_df['2+price_movement'] = (new_df['price_movement'] >= 2).astype(int)
```

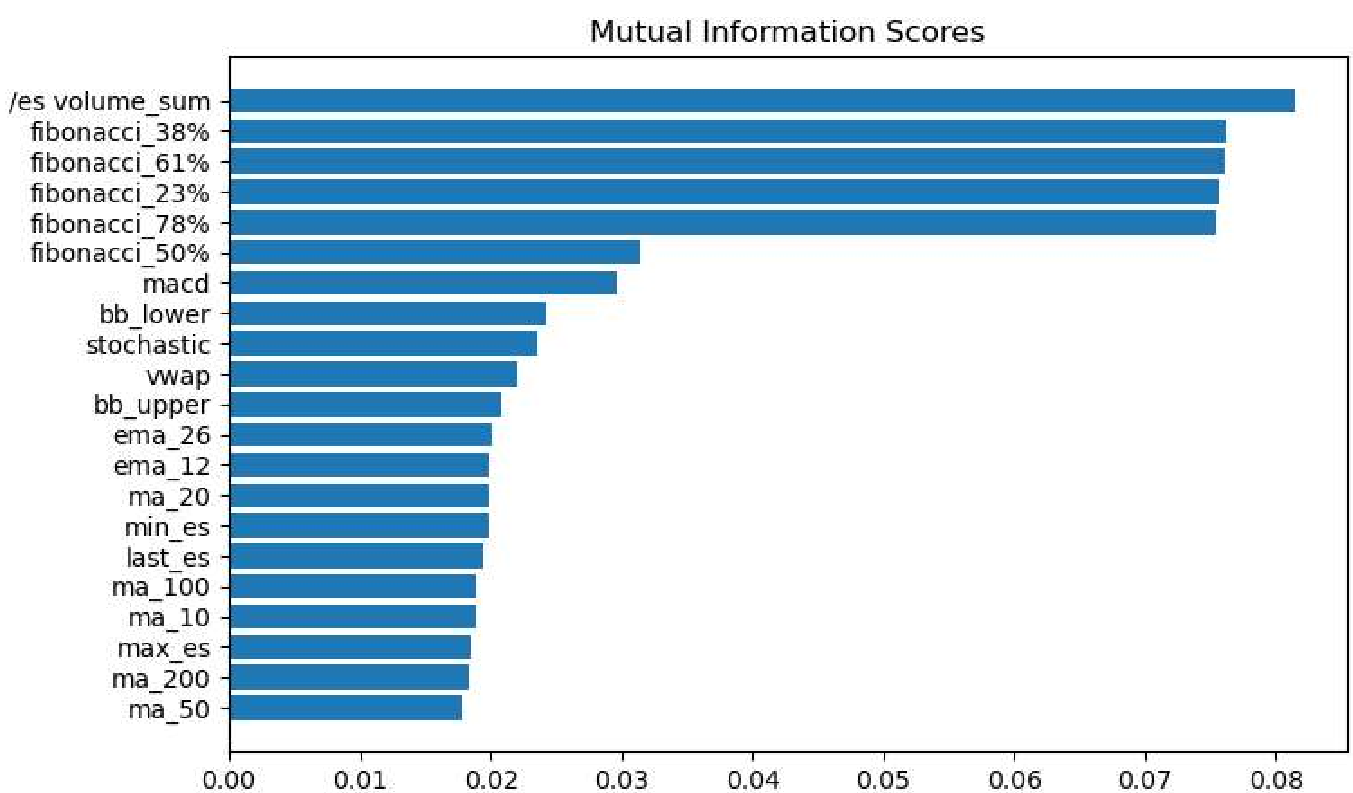
In [12]:
```
#creates a distbution graph of the prive movements.
plt.figure(figsize=(10,6))
sns.histplot(data=new_df["price_movement"], stat="probability", color='skyblue', edgecolor='black', bins=800)
plt.ylim(0, 0.25)
plt.xlim(-5,10)
plt.show()
```



In [13]:
```
#Creates a copy of the df to use for the MI
mi_x = new_df.copy()
mi_x.drop(columns="price_movement",inplace=True)
mi_y = mi_x.pop("2+price_movement")

#Label encoding for categoricals
for colname in mi_x.select_dtypes("object"):
    mi_x[colname], _ = mi_x[colname].factorize()

#All discrete features should now have integer dtypes
discrete_features = mi_x.dtypes == int
```

In [14]:
```
#Creates a function to calculate the MI scores
def make_mi_scores(mi_x, mi_y, discrete_features):
    mi_scores = mutual_info_regression(mi_x, mi_y, discrete_features=discrete_features)
    mi_scores = pd.Series(mi_scores, name="MI Scores", index=mi_x.columns)
    mi_scores = mi_scores.sort_values(ascending=False)
    return mi_scores

mi_scores = make_mi_scores(mi_x, mi_y, discrete_features)
```

In [15]:
```
#Creates a function to graph the MI scores
def plot_mi_scores(scores):
    scores = scores.sort_values(ascending=True)
    width = np.arange(len(scores))
    ticks = list(scores.index)
    plt.barh(width, scores)
    plt.yticks(width, ticks)
    plt.title("Mutual Information Scores")


plt.figure(dpi=100, figsize=(8, 5))
plot_mi_scores(mi_scores)
```



In [16]:
```
#Splits the data into x and y datasets
x=new_df.drop(columns=['price_movement','2+price_movement'])
y=new_df['2+price_movement']

#Splits the data into training and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_state=42)
```

In [17]:
```
#Logistic regression parameters
logistic_params = {'penalty': ['l1', 'l2'],
                   'C': np.logspace(-4, 4, 20),
                   'max_iter': [500,1000,1500]}

#Logistic regression grid search
logistic_model = LogisticRegression()
logistic_grid = GridSearchCV(logistic_model, logistic_params, cv=5)
logistic_grid.fit(x_train, y_train)
```

```
C:\Users\predi\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:378: FitFailedWarning:
300 fits failed out of a total of 600.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
300 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\predi\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 686, in _fit
_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\predi\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "C:\Users\predi\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 54, in _check_sol
ver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\predi\anaconda3\lib\site-packages\sklearn\model_selection\_search.py:952: UserWarning: One or more o
f the test scores are non-finite: [       nan 0.75584313        nan 0.75584313        nan 0.75584313
        nan 0.75435028        nan 0.75435028        nan 0.75435028
        nan 0.75642239        nan 0.75642239        nan 0.75642239
        nan 0.75261722        nan 0.75261722        nan 0.75261722
        nan 0.7562811         nan 0.7562811         nan 0.7562811
        nan 0.75260777        nan 0.75260777        nan 0.75260777
        nan 0.7525701         nan 0.7525701         nan 0.7525701
        nan 0.75106312        nan 0.75106312        nan 0.75106312
        nan 0.75210859        nan 0.75210859        nan 0.75210859
        nan 0.75134568        nan 0.75134568        nan 0.75134568
        nan 0.75449153        nan 0.75449153        nan 0.75449153
        nan 0.75361563        nan 0.75361563        nan 0.75361563
        nan 0.7551179         nan 0.7551179         nan 0.7551179
        nan 0.75230167        nan 0.75230167        nan 0.75230167
        nan 0.75371917        nan 0.75371917        nan 0.75371917
        nan 0.75516025        nan 0.75516025        nan 0.75516025
        nan 0.7549154         nan 0.7549154         nan 0.7549154
        nan 0.75359205        nan 0.75359205        nan 0.75359205
        nan 0.755132          nan 0.755132          nan 0.755132
        nan 0.75361563        nan 0.75361563        nan 0.75361563]
  warnings.warn(
```

Out[17]:

▸ **GridSearchCV**

▸ **estimator: LogisticRegression**

▸ LogisticRegression

In [18]:
```
#XGBoost parameters
xgb_params = {'learning_rate': [0.1, 0.2],
              'n_estimators': [100, 250, 500],
              'max_depth': [3, 5, 6],}

#XGBoost grid search
xgb_model = XGBClassifier()
xgb_grid = GridSearchCV(xgb_model, xgb_params, cv=5)
xgb_grid.fit(x_train, y_train)
```

Out[18]:

```
▸ GridSearchCV
▸ estimator: XGBClassifier
  ▸ XGBClassifier
```

In [19]:

```
#CatBoost parameters
catboost_params = {'learning_rate': [0.1, 0.2],
                   'iterations': [100, 250, 500],
                   'depth': [3, 5, 6],}

#CatBoost grid search
catboost_model = CatBoostClassifier()
catboost_grid = GridSearchCV(catboost_model, catboost_params, cv=5)
catboost_grid.fit(x_train, y_train)
```

```
492:	learn: 0.4765512	total: 8.62s	remaining: 122ms
493:	learn: 0.4765242	total: 8.64s	remaining: 105ms
494:	learn: 0.4765097	total: 8.66s	remaining: 87.4ms
495:	learn: 0.4764887	total: 8.67s	remaining: 69.9ms
496:	learn: 0.4764678	total: 8.69s	remaining: 52.4ms
497:	learn: 0.4764546	total: 8.7s	remaining: 34.9ms
498:	learn: 0.4764404	total: 8.71s	remaining: 17.5ms
499:	learn: 0.4764163	total: 8.73s	remaining: 0us
```

Out[19]:

```
▸ GridSearchCV
▸ estimator: CatBoostClassifier
  ▸ CatBoostClassifier
```

In [20]:

```python
#Creates a function to ccalulate the accuracy and precision of each model.
def evaluate_model(model, x, y):
    y_pred = model.predict(x)
    accuracy = accuracy_score(y, y_pred)
    precision = precision_score(y, y_pred)
    cm = confusion_matrix(y, y_pred)
    return accuracy, precision, cm
```

In [21]:

```python
#runs each model through the the previous created function and stores the returned variables
logistic_accuracy, logistic_precision, logistic_cm= evaluate_model(logistic_grid, x_train, y_train)
xgb_accuracy, xgb_precision, xgb_cm = evaluate_model(xgb_grid, x_train, y_train)
catboost_accuracy, catboost_precision, catboost_cm = evaluate_model(catboost_grid, x_train, y_train)

#prints the results
print("Logistic Regression Best Parameters:", logistic_grid.best_params_)
print("Logistic Regression Accuracy:", logistic_accuracy)
print("Logisti c Regression Precision:", logistic_precision)

print("XGBoost Best Parameters:", xgb_grid.best_params_)
print("XGBoost Accuracy:", xgb_accuracy)
print("XGBoost Precision:", xgb_precision)

print("CatBoost Best Parameters:", catboost_grid.best_params_)
print("CatBoost Accuracy:", catboost_accuracy)
print("CatBoost Precision:", catboost_precision)
```

```
Logistic Regression Best Parameters: {'C': 0.0006951927961775605, 'max_iter': 500, 'penalty': 'l2'}
Logistic Regression Accuracy: 0.7494572460594415
Logisti c Regression Precision: 0.6167173252279635
XGBoost Best Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 500}
XGBoost Accuracy: 0.7838261680394456
XGBoost Precision: 0.6630842216235211
CatBoost Best Parameters: {'depth': 3, 'iterations': 500, 'learning_rate': 0.2}
CatBoost Accuracy: 0.768388880255059
CatBoost Precision: 0.6173484622275306
```

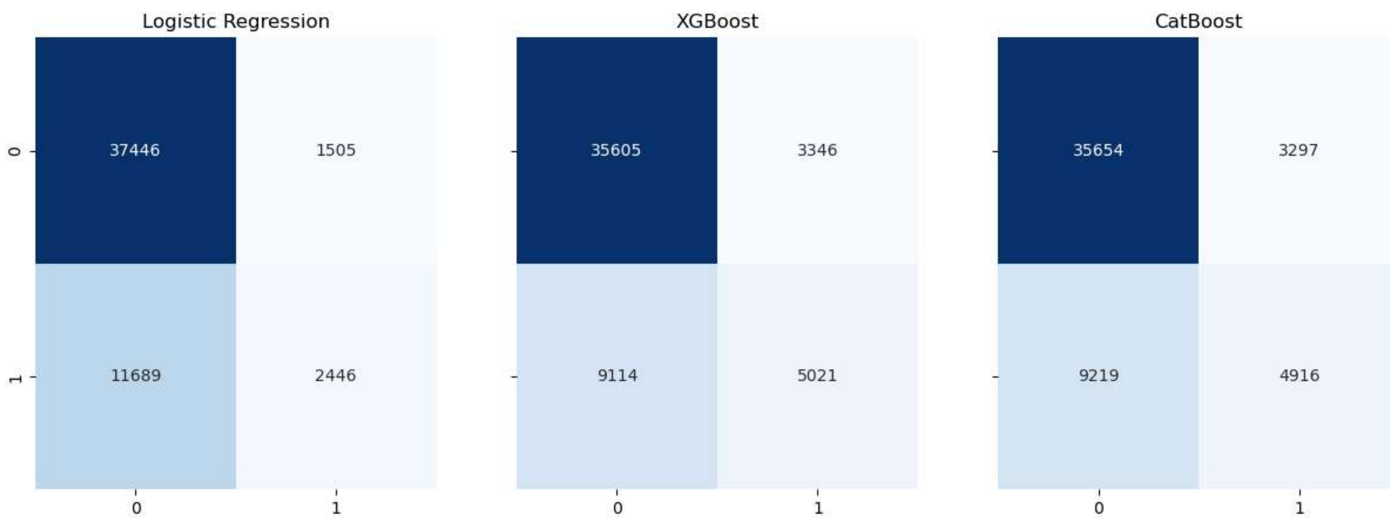
In [22]:

```python
#runs each model through the the previous create dfunction and stores the returned variables
logistic_accuracy, logistic_precision, logistic_cm  = evaluate_model(logistic_grid, x_test, y_test)
xgb_accuracy, xgb_precision, xgb_cm = evaluate_model(xgb_grid, x_test, y_test)
catboost_accuracy, catboost_precision, catboost_cm = evaluate_model(catboost_grid, x_test, y_test)

#prints the results
print("Logistic Regression Best Parameters:", logistic_grid.best_params_)
print("Logistic Regression Accuracy:", logistic_accuracy)
print("Logistic Regression Precision:", logistic_precision)

print("XGBoost Best Parameters:", xgb_grid.best_params_)
print("XGBoost Accuracy:", xgb_accuracy)
print("XGBoost Precision:", xgb_precision)

print("CatBoost Best Parameters:", catboost_grid.best_params_)
print("CatBoost Accuracy:", catboost_accuracy)
print("CatBoost Precision:", catboost_precision)
```

```
Logistic Regression Best Parameters: {'C': 0.0006951927961775605, 'max_iter': 500, 'penalty': 'l2'}
Logistic Regression Accuracy: 0.7514598952642881
Logistic Regression Precision: 0.6190837762591749
XGBoost Best Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 500}
XGBoost Accuracy: 0.7652865162189655
XGBoost Precision: 0.6000956137205689
CatBoost Best Parameters: {'depth': 3, 'iterations': 500, 'learning_rate': 0.2}
CatBoost Accuracy: 0.7642316241570282
CatBoost Precision: 0.5985632533787897
```

In [23]:

```python
#Combines all of the confuision matrixs into a single visual.
fig, axes = plt.subplots(1, 3, figsize=(15, 5), sharey=True)

sns.heatmap(logistic_cm, annot=True, fmt="d", cmap="Blues", cbar=False, ax=axes[0])
axes[0].set_title("Logistic Regression")

sns.heatmap(xgb_cm, annot=True, fmt="d", cmap="Blues", cbar=False, ax=axes[1])
axes[1].set_title("XGBoost")

# Model 3
sns.heatmap(catboost_cm, annot=True, fmt="d", cmap="Blues", cbar=False, ax=axes[2])
axes[2].set_title("CatBoost")

plt.show()
```



In [ ]: