```python
# modules.project_helper.py
# https://github.com/QuantCS109/TrumpTweets/blob/master/modules/project_helper.py
# This is a .py file used in the notebooks


import pandas as pd
import numpy as np
from collections import Counter
from pytz import timezone
from datetime import timedelta
import pickle
import re
from sklearn.ensemble import RandomForestClassifier



class TweetData:

    def __init__(self, file='../data/input_data/trump_archive_db.csv'):
        self.file = file
        self.raw_data = []
        self.error_tweets = {}

        self.read_data()
        self.raw_tweets = self.parse()
        self.clean_tweets = self.clean()
        self.text = self.create_text()
        self.words = self.tokenize_text()
        self.vocab_to_int, self.int_to_vocab = self.create_lookup_tables()
        self.int_words = self.create_int_words()

        self.daily_tweets = None
        self.get_daily_tweets()

    def read_data(self):
        with open(self.file, mode='r',errors='ignore') as f:
            for row in f:
                self.raw_data.append(row)

    def parse(self):
        timestamps = []
        tweets = []
        raw_tweets = pd.DataFrame(columns=['tweets'])
        for i, tweet in enumerate(self.raw_data):
            try:
                timestamps.append(timezone('US/Central').localize(pd.to_datetime((tweet
                [-21:-2]))))
                tweets.append(tweet[:-22])
            except:
                self.error_tweets[i] = tweet
        raw_tweets['tweets'] = tweets
        raw_tweets.index = timestamps
        raw_tweets.index.name = 'timestamp'

        return raw_tweets

    def clean_step_1(self, tweet):

        # Remove whitespace before and after tweet
        tweet = tweet.strip(' ')
        tweet = tweet.lstrip('\"')
        tweet = tweet + '\n\n'
        return tweet

    def clean_step_2(self, tweet):
        # Remove http links
        tweet = re.sub(r"http\S+", '', tweet)
```

```
67          # Remove hash tags
68          tweet = re.sub(r"#\S+", '', tweet)
69          # Remove twitter handles
70          tweet = re.sub(r"@\S+", '', tweet)
71          # Turn everything to lower case
72          tweet = tweet.lower()
73          # Remove symbols other than letters in the alphabet and numbers
74          tweet = re.sub(r"\'", '', tweet)
75          tweet = re.sub(r'[^a-zA-Z0-9]', ' ', tweet)
76          # Remove whitespace before and after tweet, add one white space
77          tweet = re.sub(r'[^a-zA-Z]', ' ', tweet)
78          tweet = ' '.join(tweet.split())
79          tweet = tweet + ' '
80          return tweet
81
82      def clean(self):
83          clean_tweets = pd.DataFrame(columns=['tweets'])
84          clean_tweets['tweets'] = self.raw_tweets['tweets'].apply(self.clean_step_1)
85          clean_tweets['tweets'] = clean_tweets['tweets'].apply(self.clean_step_2)
86          clean_tweets.index = self.raw_tweets.index
87          return clean_tweets
88
89      def create_text(self):
90          text = ''.join(self.clean_tweets.tweets)
91          return text
92
93      def tokenize_text(self):
94          words = self.text.split()
95          # Remove all words with  5 or fewer occurrences
96          word_counts = Counter(words)
97          return [word for word in words if word_counts[word] > 5]
98
99      def create_lookup_tables(self):
100
101         word_counts = Counter(self.words)
102         # words sorted in descending frequency
103         sorted_vocab = sorted(word_counts, key=word_counts.get, reverse=True)
104         int_to_vocab = {ii: word for ii, word in enumerate(sorted_vocab)}
105         vocab_to_int = {word: ii for ii, word in int_to_vocab.items()}
106         return vocab_to_int, int_to_vocab
107
108     def create_int_words(self):
109         return [self.vocab_to_int[word] for word in self.words]
110
111     def get_daily_tweets(self):
112         self.clean_tweets['timestamp'] = self.clean_tweets.index
113         after_4_tweets = self.clean_tweets.timestamp.dt.hour >= 15
114         self.clean_tweets['after4_date'] = self.clean_tweets.timestamp.dt.date
115         self.clean_tweets.loc[after_4_tweets, 'after4_date'] = self.clean_tweets.
            timestamp[after_4_tweets].dt.date\
116                                                                 + timedelta(days=1)
117         self.daily_tweets = self.clean_tweets.groupby('after4_date')['tweets'].apply(
            lambda x: ' '.join(x))
118         self.daily_tweets = self.daily_tweets.to_frame('tweets')
119         self.daily_tweets.index.name = 'date'
120
121
122 class APIData(TweetData):
123
124     def __init__(self, file='../data/input_data/trumptwits.csv'):
125         super().__init__(file=file)
126
127     def read_data(self):
128         self.raw_data = pd.read_csv(self.file)
129
130     def parse(self):
131         raw_tweets = pd.DataFrame(columns=['tweets'])
```

```
132             raw_tweets['tweets'] = self.raw_data['text']
133             raw_tweets.index = pd.to_datetime(self.raw_data['time'])
134             raw_tweets.index.name = 'timestamp'
135             raw_tweets = raw_tweets.sort_index()
136             raw_tweets = raw_tweets.tz_convert('US/Central')
137             return raw_tweets
138
139         def clean_step_1(self, tweet):
140             tweet = tweet.lstrip('b\'')
141             tweet.rstrip('\n\n\n\n')
142             tweet.rstrip('\'')
143             return tweet
144
145
146     class IntradayData:
147
148         def __init__(self,file='../data/input_data/ES_intraday.csv'):
149             self.file = file
150             self.raw_data = self.read_data()
151
152         def read_data(self):
153             fin_data = pd.read_csv(self.file)
154             fin_data.index = pd.to_datetime(fin_data['Date'] + ' ' + fin_data['Time']).dt.
            tz_localize('US/Central')
155             fin_data.index.name = 'timestamp'
156             fin_data = fin_data.drop(columns=['Date', 'Time'])
157             return fin_data
158
159         def get_data(self):
160             return self.raw_data[['Open', 'Close']]
161
162
163     class FuturesCloseData:
164         def __init__(self, path='../data/input_data/futures_close.csv'):
165             self.instrument_list = ['ES', 'NQ', 'CD', 'EC', 'JY', 'MP', 'TY', 'US', 'C', 'S'
            , 'W', 'CL', 'GC']
166             self.df = self.load(path)
167
168         def load(self, path):
169             df = pd.read_csv(path)
170             df.set_index('date', inplace=True)
171             df.index = pd.to_datetime(df.index)
172             return df
173
174         def features(self, inst):
175             return self.momentum(inst)
176
177         def price(self, inst):
178             return self.df[inst]
179
180         def returns(self, inst, start=0, end=1):
181             returns = (self.df[inst].shift(-end) - self.df[inst].shift(-start)) / self.df[
            inst].shift(-start)
182             return returns
183
184         def momentum(self, inst, lag=60):
185             momo = pd.DataFrame((self.df[inst] - self.df[inst].shift(lag)) / self.df[inst])
186             momo = momo.dropna()
187             momo.columns += '_{}D'.format(lag)
188             return momo
189
190         def log_returns(self, start=0, end=1):
191              return np.log(self.df.shift(-end)) - np.log(self.df.shift(-start))
192
193         def single_log_returns(self, inst,  start=0, end=1):
194              return np.log(self.df[inst].shift(-end)) - np.log(self.df[inst].shift(-start))
195
```

```
196
197 class VolFeatures:
198     def __init__(self, path='../data/features/vol_features.pkl'):
199         self.instrument_list = ['ES', 'NQ', 'CD', 'EC', 'JY', 'MP', 'TY', 'US', 'C', 'S'
        , 'W', 'CL', 'GC']
200         self.df = self.load(path)
201         self.col_dict = {inst: [key for key in self.df.columns if re.match(r"{}_+".
        format(inst), key)]
202                          for inst in self.instrument_list}
203
204     def features(self, inst):
205         return self.df[self.col_dict[inst]]
206
207     def load(self, path):
208         pickle_in = open(path, "rb")
209         vol_pd = pickle.load(pickle_in)
210         pickle_in.close()
211         return vol_pd.fillna(vol_pd.mean())
212
213
214 class TweetReturnsFeatures(VolFeatures):
215     def __init__(self, path='../data/features/tweet_returns_features.csv'):
216         super().__init__(path)
217
218     def load(self, path):
219         tweet_returns = pd.read_csv(path)
220         tweet_returns.set_index('date', inplace=True)
221         tweet_returns.index = pd.to_datetime(tweet_returns.index)
222         return tweet_returns
223
224 class MarketFeatures    (TweetReturnsFeatures):
225     def __init__(self, path='../data/features/market_features.csv'):
226         super().__init__(path)
227
228
229 class TradeModel:
230
231     def __init__(self, model=RandomForestClassifier, *args, **kwargs):
232         self.model = model(*args, **kwargs)
233
234     def fit(self, X, y):
235         self.model.fit(X, y)
236
237     def position(self, X, cutoff=0.55):
238         # converting predictions from {0,1} to {-1,1}, short/long
239         position = 2 * self.model.predict(X) - 1
240         position[self.model.predict_proba(X).max(axis=1) <= cutoff] = 0
241         return position
242
243     def _strategy_returns(self, x, y):
244         strat_rets = x[:-2] * y[:-2]
245         strat_rets_cum =  strat_rets.cumsum()
246         return strat_rets, strat_rets_cum
247
248     def strategy_returns(self, X, returns, cutoff=0.55):
249         return self._strategy_returns(returns, self.position(X, cutoff))
250
251     def sharpe(self, X, returns, cutoff=0.55):
252         rets = self.strategy_returns(X, returns, cutoff)[0]
253         return np.mean(rets) / np.std(rets)
254
255
256
257
258
259
260
```