Part II: Prediction

1. Download & pre-process the data :

Run a python script, in which includes the following major functions:

a.Data download. Parameters may passed to the function to download whatever file requested.

```
def get_data_from_url(quarter):
    print('downloading...')
    urllib.request.urlretrieve('https://freddiemac.embs.com/FLoan/Data/historical_data1_' + str(quarter) + '.zip',
                               DIR_NAME + str(quarter) + '.zip')
    # unzip_files(year)
    try:
        zip_ref = zipfile.ZipFile(DIR_NAME + str(quarter) + '.zip', 'r')
        zip_ref.extractall(DIR_NAME)
        zip_ref.close()

        write_into_csv(quarter)
    except zipfile.BadZipfile:
        print (zipfile.BadZipfile)
```

b. Data cleaning. The cleansed data is written to a csv file with renamed columns.

```
def data_cleaning():
    fileList = fileList = glob.glob('data/*.csv')
    for file in fileList:
        df = pd.read_csv(file,error_bad_lines=False)
        print('categorical cleaning...')
        for col in ['C','G','H','M','N','O','P','Q','T','U','V','W','X']:
            mode = pd.DataFrame(df.groupby(col).size().rename('cnt')).idxmax()[0]
            df[col] = df[col].fillna(mode)
        print('numerical cleaning...')
        for col in ['E','F','I','J','K','L','R']:
            dfmean = df[(df[col] != 999)|(df[col] != None)]
            mean = int(dfmean[col].mean(axis=0))
            df[col] = df[col].fillna(mean)

        filename = "%sclean.csv"%file[:-4]
        df.to_csv(filename)

        print('cleaning finished.')
```

c. Data processing. The columns are processed as dummy values to be used against prediction.

d. Main function as the entrance point. In this function the program logs in and and uses the defined function.

Please note that all output files are put into a data/ directory.

```python
def start_execution():
    with requests.Session() as sess:
        sess.get(login_page_url);
        php_session_cookie = sess.cookies['PHPSESSID']
        login_payload = {'username' : USERNAME, 'password' : PASSWORD,'cookie':php_session_cookie}
        sess.post(login_page_url, data = login_payload)
        download_page_payload = {'accept': 'Yes', 'action': 'acceptTandC', 'acceptSubmit': 'Continue', 'cookie': php_session_cookie}
        sess.post(download_page_url, data=download_page_payload)
        create_directory(DIR_NAME)
#       create_csv()

        get_data_from_url(START)
        get_data_from_url("Q12003")
        get_data_from_url("Q12005")
        get_data_from_url("Q12007")
        get_data_from_url("Q12009")
        get_data_from_url(END)

        get_data_from_url("Q22005")
        get_data_from_url("Q32005")
        get_data_from_url("Q42005")

        get_data_from_url("Q22007")
        get_data_from_url("Q32007")
        get_data_from_url("Q42007")

        data_cleaning()
        preprocessing()
```

2. Local Analysis

According to:

```python
selector = SelectKBest(f_regression, k=20).fit(X_train,y_train)
k_best_features = X_train.columns.values[selector.get_support()]
```

The program can select the best features for further analysis:

```python
k_best_features = ['D', 'F', 'I', 'K', 'L', 'U', 'C_Y', 'H_I', 'H_P', 'Q_MH', 'T_P',
 'W_BANKOFAMERICA_NA', 'W_COUNTRYWIDE', 'W_GMACMTGECORP', 'W_NATLCITYMTGECO',
 'W_WASHINGTONMUTUALBANK', 'X_BANKOFAMERICA_NA', 'X_COUNTRYWIDE',
 'X_GMACMORTGAGE_LLC', 'X_NATLCITYMTGECO']
```

These features are used against the following required algorithms and functions.

```python
#Linear, random forestk, and neural network models
from sklearn.linear_model import LinearRegression

linear = LinearRegression()
linear.fit(X_train,y_train)
linearPredict = linear.predict(X_test)
printPerformance(linearPredict)

from sklearn.ensemble import RandomForestRegressor
randomForest = RandomForestRegressor(max_depth= 18, n_estimators = 16, random_state=2)
randomForest.fit(X_train,y_train)
randomForestPredict = randomForest.predict(X_test)
printPerformance(randomForestPredict)

from sklearn.neural_network import MLPRegressor
neuralNetwork = MLPRegressor()
neuralNetwork.fit(X_train,y_train)
neuralNetworkPredict = neuralNetwork.predict(X_test)
printPerformance(neuralNetworkPredict)

# Build RF classifier to use in feature selection
from sklearn.ensemble import RandomForestClassifier
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
clf = RandomForestClassifier(n_estimators=100, n_jobs=-1)
```

For each algorithm, the program can also provide stepwise and exhaustive feature selection approaches.

```
#Forward Selection
fs = sfs(clf,
        k_features=9,
        forward=False,
        floating=False,
        n_jobs=-1,
        verbose=2,
        scoring='neg_mean_absolute_error',
        cv=10)
fs.fit(X_train,y_train)
print('Best MAE score: %.2f' % fs.best_score_ * (-1))
print('Best subset:', fs.best_feature_names_)
```

```
#Backward Selection
bs = sfs(clf,
        k_features=9,
        forward=False,
        floating=False,
        n_jobs=-1,
        verbose=2,
        scoring='neg_mean_absolute_error',
        cv=10)
bs.fit(X_train,y_train)
```

```
#Exhaustive Selection

from mlxtend.feature_selection import ExhaustiveFeatureSelector as efs
es = efs(clf,
          min_features=8,
          max_features=11,
          scoring='neg_mean_absolute_error',
          n_jobs=-1,
          print_progress=True,
          cv=8)
es.fit(X_train,y_train_encoded)
print('Best MAE score: %.2f' % efs.best_score_ * (-1))
print('Best subset:', efs.best_feature_names_)
```

…and then validates against each algorithm.

```
#Validation
from sklearn.model_selection import cross_val_score
scores = cross_val_score(linear, X_test, y_test, cv=3)
print(scores)
scores = cross_val_score(randomForest, X_test, y_test, cv=3)
print(scores)
scores = cross_val_score(neuralNetwork, X_test, y_test, cv=3)
print(scores)
```

3. AutoML Usages

a.Run a TPOT python script like:

```
#tpot
from tpot import TPOTRegressor

pipeline_optimizer = TPOTRegressor(generations=5, population_size=20, cv=5,
                                    random_state=42, verbosity=2)
pipeline_optimizer.fit(X_train, y_train)
print(pipeline_optimizer.score(X_test, y_test))
pipeline_optimizer.export('tpot_exported_pipeline.py')
```

…and it gives the following pipeline:

```
#Pipeline Exported
from sklearn.linear_model import LassoLarsCV
from sklearn.model_selection import train_test_split

# NOTE: Make sure that the class is labeled 'target' in the data file
tpot_data = df
features = tpot_data.drop('0', axis=1).values
training_features, testing_features, training_target, testing_target = \
            train_test_split(features, tpot_data['0'].values, random_state=42)

# Average CV score on the training set was:-6.254833533625527e-26
exported_pipeline = LassoLarsCV(normalize=True)

exported_pipeline.fit(training_features, training_target)
results = exported_pipeline.predict(testing_features)
```

The program checks its performance by running the following function.

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error
from math import sqrt
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

def printPerformance(pred):
    print(pred)
    print("RMSE: %.2f"
          % sqrt(mean_squared_error(y_test, pred)))
    print("MAPE: %.2f"
          % mean_absolute_percentage_error(y_test, pred)+'%')
    print("MAE: %.2f"
          % mean_absolute_error(y_test, pred))
```

b. Run a H2o jupyter notebook. Please note that this step requires your credentials after mounting driverless AI in docker.

```python
import h2o
from h2o.automl import H2OAutoML
h2o.init()

h2oq12005 = h2o.import_file("data/Q12005.csv")
h2oq22005 = h2o.import_file("data/Q22005.csv")

X = h2oq22005.columns.remove("0")
y = "0"
h2oML = H2OAutoML(max_models = 30, max_runtime_secs=300, seed = 1)
h2oML.train(x= x, y = y, training_frame= h2oq12005, leaderboard_frame= h2oq22005)

aml.leaderboard.as_data_frame()
```

4. What-If analysis:

For this part, read different data frames as training sets and see the performance of each.

…which gives the result of:

```
fitting...
fitted.

RMSE: 0.35
MAPE: 4.32%
MAE: 0.26
Two Years Later..

RMSE: 2.08
MAPE: 42.23%
MAE: 2.05
```

```python
#What if there is a financial crisis...

df_train = pd.concat([df7,df72,df73,df74],axis = 0).sample(frac = 0.1)
df_test = pd.concat([df8,df72,df73,df74],axis = 0).sample(frac = 0.1)

#Pipline random forest
from sklearn.ensemble import RandomForestRegressor
randomForest = RandomForestRegressor(max_depth= 18, n_estimators = 16, random_state=2)

X_train = df_train[k_best_features]
y_train = df_train['0']
X_test = df_test[k_best_features]
y_test = df_test['0']

randomForest.fit(X_train,y_train)
randomForest.predict(X_test)

#Two Years Later..
X_test = df9[k_best_features]
y_test = df9['0']
randomForest.fit(X_train,y_train)
printPerformance(randomForest.predict(X_test))
```

As for

```
#What if there is a economy boom...

df_train = pd.concat([df0,df2,df4,df6,df8,df10,df12],axis = 0).sample(frac = 0.1)

X_train = df_train[k_best_features].head(shape)
y_train = df_train['0'].tail(df_test.shape)
X_test = df12
randomForest.fit(X_train,y_train)
printPerformance(randomForest.predict(X_test))
```

It shows:

```
RMSE: 3.64
MAPE: 98.38%
MAE: 3.57
```

Thus, the program does not recommend this model by introducing low performance.

And the performance against regime change…

```
#What if there is a regime change from election

X_train = df6[k_best_features].head(shape)
y_train = df6['0'].tail(df_test.shape)
X_test = df12[k_best_features]
y_test = df12['0']

randomForest.fit(X_train,y_train)
printPerformance(randomForest.predict(X_test))
```

```
RMSE: 2.32
MAPE: 62.49%
MAE: 2.28
```