Buy-Rent-Sell, Real Estate Property Investments

Personal Investment Opportunity Identifier

Syracuse Applied Data Science, IST-707 Data Analytics

Ryan Timbrook (RTIMBROO)
DATE: 9/8/2019 ASSIGNMENT: Final Project

1. Introduction

Targeting low risk property investment opportunities for property management firms or individual investors who buy-rent-sell single family homes throughout the United States.

Given a base set of investment criteria, provide a predicted N-best list of US geolocation regions by zip code that offer the best ROI.

Problem Statement:

- How to predict a low risk / high yield return on property investment in a volatile market.
- Buy low, rent fair, sell high…
- Where and when to buy and sell that maximizes investment profits.
- Forecast future growth and decline of a region that yields Net Present Value (NPE) measurements significant enough to act on.

Base Real Estate data provided by: [Zillow (files.zillowstatic.com/research/public/Zip/Zip_Zhvi_SingleFamilyResidence.csv)](files.zillowstatic.com/research/public/Zip/Zip_Zhvi_SingleFamilyResidence.csv)

Base Federal Reserve Interest Rates data provided by: [kaggle (https://www.kaggle.com/federalreserve/interest-rates)](https://www.kaggle.com/federalreserve/interest-rates)

==

1.1 About the Data

Dataset Info

Coding Environment Setup

Import packages

In [1]:
```
# toggle for working with colab
isColab = False
```

ONLY RUN WHEN WORKING ON COLAB

In [ ]:
```
# mount google drive for working in colab
'''
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)

# working within colab, set base working directory
base_dir = "./gdrive/My Drive/IST707_PRJ_Realestate/buy_rent_sell/"

# validate directory mapping
#ls f'{base_dir}'

# upload custome python files
from google.colab import files
uploaded_files = files.upload()

# print files uploaded
for f in uploaded_files.keys():
  print(f'file name: {f}')

isColab = True
'''
```

In [5]:
```
# import packages
import pandas as pd                               # data frame operations
import numpy as np                             # arrays and match functions
import random
import time
import gc
import os
import pickle
from pathlib import Path

import seaborn as sns                             # uses for visualizations
import matplotlib.pyplot as plt                   # used for 2D plotting
%matplotlib inline
plt.style.use('fivethirtyeight')

import warnings
from timeit import default_timer                  # performance processing time
import logging                                    # logging framework
# documentation can be found: https://uszipcode.readthedocs.io/index.html
!pip install uszipcode
import uszipcode # programmable zipcode database
from tqdm.autonotebook import tqdm
```

```
C:\ProgramData\Anaconda3\lib\site-packages\tqdm\autonotebook\__init__.py:1
4: TqdmExperimentalWarning: Using `tqdm.autonotebook.tqdm` in notebook mod
e. Use `tqdm.tqdm` instead to force console mode (e.g. in jupyter console)
  " (e.g. in jupyter console)", TqdmExperimentalWarning)

Requirement already satisfied: uszipcode in c:\programdata\anaconda3\lib\si
te-packages (0.2.2)
Requirement already satisfied: sqlalchemy in c:\programdata\anaconda3\lib\s
ite-packages (from uszipcode) (1.3.1)
Requirement already satisfied: requests in c:\programdata\anaconda3\lib\sit
e-packages (from uszipcode) (2.21.0)
Requirement already satisfied: pathlib-mate in c:\programdata\anaconda3\lib
\site-packages (from uszipcode) (0.0.15)
Requirement already satisfied: attrs in c:\programdata\anaconda3\lib\site-p
ackages (from uszipcode) (19.1.0)
Requirement already satisfied: idna<2.9,>=2.5 in c:\programdata\anaconda3\l
ib\site-packages (from requests->uszipcode) (2.8)
Requirement already satisfied: certifi>=2017.4.17 in c:\programdata\anacond
a3\lib\site-packages (from requests->uszipcode) (2019.3.9)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in c:\programdata\anac
onda3\lib\site-packages (from requests->uszipcode) (3.0.4)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in c:\programdata\anac
onda3\lib\site-packages (from requests->uszipcode) (1.24.1)
Requirement already satisfied: autopep8 in c:\programdata\anaconda3\lib\sit
e-packages (from pathlib-mate->uszipcode) (1.4.4)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-pac
kages (from pathlib-mate->uszipcode) (1.12.0)
Requirement already satisfied: pycodestyle>=2.4.0 in c:\programdata\anacond
a3\lib\site-packages (from autopep8->pathlib-mate->uszipcode) (2.5.0)
```

In [6]:
```
# custome python packages
import rtimbroo_ist_utils as rt                # custome python helper functi
import brs_utils as brs # custome functions specific to buy_sell_rent project
```

In [7]:
```
#from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
```

Time series models at scale. Based on the research from facebook - [Prophet (https://research.fb.com/prophet-forecasting-at-scale/)](https://research.fb.com/prophet-forecasting-at-scale/) - allows the user to quickly produce high quality forecasts with the ability to adjust multiple parameters. Initial code modeled after Digital Ocean's [tutorial (https://www.digitalocean.com/community/tutorials/a-guide-to-time-series-forecasting-with-prophet-in-python-3)](https://www.digitalocean.com/community/tutorials/a-guide-to-time-series-forecasting-with-prophet-in-python-3).
Data Tranansformations: Prophet requires columns to be in certain formats Use python transpose

*[Prophet Quick Start Guide: (https://facebook.github.io/prophet/docs/quick_start.html#python-api)](https://facebook.github.io/prophet/docs/quick_start.html#python-api)

In [8]:
```
# timeseries packages
from fbprophet import Prophet
```

```
ERROR:fbprophet:Importing plotly failed. Interactive plots will not work.
```

In [9]:
```
# set global properties
if not isColab:
    dataDir = './data'
    outputDir = './output'
    configDir = './config'
    logOutDir = './logs'
    imageDir = './images'
    modelDir = './models'
else:
    # working within colab
    dataDir = f'{base_dir}data'
    outputDir = f'{base_dir}output'
    configDir = f'{base_dir}config'
    logOutDir = f'{base_dir}logs'
    imageDir = f'{base_dir}images'
    modelDir = f'{base_dir}models'

modelPerformance = {}
modelNames = 'zip_time_series'
appName = 'rt_brs_time_series'
loglevel = 10 # 10-DEBUG, 20-INFO, 30-WARNING, 40-ERROR, 50-CRITICAL

# focus in on a single state and selected set of regions for this initial pro
focus_state = 'WA'
regions = []

# time series training set years
ts_col_years = [str(y) for y in range(2000,2019)]
ts_train_years = [str(y) for y in range(2000,2018)]
ts_validate_year = '2018'

# sub directory for storing models
trainDir = 'train'
future5Dir = 'future_5'
# time series model projection time
ts_pred_periods = 12*5
```

In [10]:
```
# get a logger for troubleshooting / data exploration
logger = rt.getFileLogger(logOutDir,appName,level=loglevel)
np.random.seed(42) # NumPy
```

In [11]:
```
# create base output directories if they don't exist
if not os.path.exists(outputDir): os.mkdir(outputDir)
if not os.path.exists(logOutDir): os.mkdir(logOutDir)
if not os.path.exists(imageDir): os.mkdir(imageDir)
if not os.path.exists(modelDir): os.mkdir(modelDir)
```

In [ ]:

1.3 Obtain the data

- Using the base data available from [Zillow (files.zillowstatic.com/research/public/Zip/Zip_Zhvi_SingleFamilyResidence.csv)](files.zillowstatic.com/research/public/Zip/Zip_Zhvi_SingleFamilyResidence.csv)

Zillow Home Value Index (ZHVI): A smoothed, seassonally adjusted measure of the median estimated home value across a given region and housing type. It is a dollar-denominated [alternative to repeat-sales indices (https://wp.zillowstatic.com/3/ZHVI-InfoSheet-04ed2b.pdf)](https://wp.zillowstatic.com/3/ZHVI-InfoSheet-04ed2b.pdf).

OBTAIN Interest Rates data from Kaggel

- Using the dataset provided by the kaggel [Federal Reserve Interest Rates (https://www.kaggle.com/federalreserve/interest-rates/downloads/interest-rates.zip/1)](https://www.kaggle.com/federalreserve/interest-rates/downloads/interest-rates.zip/1)

In [12]:

```
# data files to load
zip_zillow_sfr_file = 'Zip_Zhvi_SingleFamilyResidence.csv'
zip_zillow_all_homes_file = 'Zip_Zhvi_AllHomes.csv'
zip_zillow_rpsf_sfr_file = 'Zip_MedianRentalPricePerSqft_Sfr.csv'
zip_zillow_rp_all_homes_file = 'Zip_MedianRentalPrice_AllHomes.csv'
zip_zillow_lp_all_homes_file = 'Zip_MedianListingPrice_AllHomes.csv'

# interest rate data set - kaggel
interest_rates_file = 'interest_rates_kaggel.csv'

# economic datasets - https://datahub.io/core
interest_rates_dh = 'interest_rates.csv'
inflation_consumer = 'inflation-consumer.csv'
inflation_gdp = 'inflation-gdp.csv'
education_budget = 'education_budget_data.csv'
population = 'population.csv'
investor_flow_monthly = 'investor_flow_funds_monthly.csv'
housing_price_cities = 'housing_price_cities.csv'
household_income = 'household-income.csv'
employment = 'employment.csv'
cpi = 'cpi.csv'
cash_surp_def = 'cash-surp-def_csv.csv'
bonds_yeilds_10y = 'bonds_yields_10y.csv'
gdp_quarter = 'gdp_quarter.csv'
gdp_year = 'gdp_year.csv'
```

In [13]:

```
%%time
zip_zillow_sfr = pd.read_csv(dataDir+'/'+zip_zillow_sfr_file, error_bad_lines
zip_zillow_all = pd.read_csv(dataDir+'/'+zip_zillow_all_homes_file, error_bad
zip_zillow_rpsf_sfr = pd.read_csv(dataDir+'/'+zip_zillow_rpsf_sfr_file, error
zip_zillow_rp_all = pd.read_csv(dataDir+'/'+zip_zillow_rp_all_homes_file, err
zip_zillow_lp_all = pd.read_csv(dataDir+'/'+zip_zillow_lp_all_homes_file, err

re_datasets = {'Single_Family_Residence':zip_zillow_sfr,'All_Homes':zip_zillo
            'RentalPrice_PSF':zip_zillow_rpsf_sfr,'RentalPrice_All_Homes':zip
           'ListingPrice_All_Homes':zip_zillow_lp_all}

# dataset from kaggle
interest_rates = pd.read_csv(f'{dataDir}/{interest_rates_file}',error_bad_lin

# economic data from datahub.io/core
interest_rates_dh = pd.read_csv(f'{dataDir}/{interest_rates_dh}',error_bad_li
inflation_consumer = pd.read_csv(f'{dataDir}/{inflation_consumer}',error_bad_
inflation_gdp = pd.read_csv(f'{dataDir}/{inflation_gdp}',error_bad_lines=Fals
education_budget = pd.read_csv(f'{dataDir}/{education_budget}',error_bad_line
population = pd.read_csv(f'{dataDir}/{population}',error_bad_lines=False, enc
investor_flow_monthly = pd.read_csv(f'{dataDir}/{investor_flow_monthly}',erro
housing_price_cities = pd.read_csv(f'{dataDir}/{housing_price_cities}',error_
household_income = pd.read_csv(f'{dataDir}/{household_income}',error_bad_line
employment = pd.read_csv(f'{dataDir}/{employment}',error_bad_lines=False, enc
cpi = pd.read_csv(f'{dataDir}/{cpi}',error_bad_lines=False, encoding = "ISO-8
cash_surp_def = pd.read_csv(f'{dataDir}/{cash_surp_def}',error_bad_lines=Fals
bonds_yeilds_10y = pd.read_csv(f'{dataDir}/{bonds_yeilds_10y}',error_bad_line
gdp_quarter = pd.read_csv(f'{dataDir}/{gdp_quarter}',error_bad_lines=False, e
gdp_year = pd.read_csv(f'{dataDir}/{gdp_year}',error_bad_lines=False, encodin
```

```
Wall time: 2.34 s
```

In [14]:

```
# REAL ESTATE DATASET
# look over the datasets
for k,v in re_datasets.items():
    logger.info(f'{k} shape: {v.shape}')
    logger.info(f'{k} memory usage: {rt.mem_usage(v)}')
    logger.info(f'{k} info: {v.info()}')
    logger.info(f'{k} NaN Count: {rt.getNaNCount(v)}')
    rt.findColumnsNaN(v,logger,rowIndex=False)
    print('')
```

```
INFO:file_logger:Single_Family_Residence shape: (15752, 287)
INFO:file_logger:Single_Family_Residence memory usage: 38.03 MB

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15752 entries, 0 to 15751
Columns: 287 entries, RegionID to 2019-07
dtypes: float64(231), int64(52), object(4)
memory usage: 34.5+ MB

INFO:file_logger:Single_Family_Residence info: None
INFO:file_logger:Single_Family_Residence NaN Count: (189753, 2399)



INFO:file_logger:All_Homes shape: (15845, 287)
INFO:file_logger:All_Homes memory usage: 38.25 MB

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15845 entries, 0 to 15844
Columns: 287 entries, RegionID to 2019-07
dtypes: float64(231), int64(52), object(4)
memory usage: 34.7+ MB

INFO:file_logger:All_Homes info: None
INFO:file_logger:All_Homes NaN Count: (190696, 2408)



INFO:file_logger:RentalPrice_PSF shape: (2626, 120)
INFO:file_logger:RentalPrice_PSF memory usage: 3.02 MB

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2626 entries, 0 to 2625
Columns: 120 entries, RegionName to 2019-07
dtypes: float64(114), int64(2), object(4)
memory usage: 2.4+ MB

INFO:file_logger:RentalPrice_PSF info: None
INFO:file_logger:RentalPrice_PSF NaN Count: (165287, 2622)



INFO:file_logger:RentalPrice_All_Homes shape: (3453, 120)
INFO:file_logger:RentalPrice_All_Homes memory usage: 3.97 MB

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3453 entries, 0 to 3452
Columns: 120 entries, RegionName to 2019-07
```

```
dtypes: float64(114), int64(2), object(4)
memory usage: 3.2+ MB

INFO:file_logger:RentalPrice_All_Homes info: None
INFO:file_logger:RentalPrice_All_Homes NaN Count: (215022, 3448)


INFO:file_logger:ListingPrice_All_Homes shape: (10839, 121)
INFO:file_logger:ListingPrice_All_Homes memory usage: 12.47 MB

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10839 entries, 0 to 10838
Columns: 121 entries, RegionName to 2019-07
dtypes: float64(115), int64(2), object(4)
memory usage: 10.0+ MB

INFO:file_logger:ListingPrice_All_Homes info: None
INFO:file_logger:ListingPrice_All_Homes NaN Count: (372530, 5488)
```

In [ ]:

In [15]:

```
# quick look at interest rates
logger.info(f'interest_rates shape: {interest_rates.shape}')
logger.info(f'interest_rates memory usage: {rt.mem_usage(interest_rates)}')
logger.info(f'interest_rates info: {interest_rates.info()}')
logger.info(f'interest_rates NaN Count: {rt.getNaNCount(interest_rates)}')
rt.findColumnsNaN(interest_rates,logger,rowIndex=False)
logger.info(f'interest_rates head: {interest_rates.head()}')
```

```
INFO:file_logger:interest_rates shape: (904, 10)
INFO:file_logger:interest_rates memory usage: 0.07 MB

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 904 entries, 0 to 903
Data columns (total 10 columns):
Year                            904 non-null int64
Month                           904 non-null int64
Day                             904 non-null int64
Federal Funds Target Rate       462 non-null float64
Federal Funds Upper Target      103 non-null float64
Federal Funds Lower Target      103 non-null float64
Effective Federal Funds Rate    752 non-null float64
Real GDP (Percent Change)       250 non-null float64
Unemployment Rate               752 non-null float64
Inflation Rate                  710 non-null float64
dtypes: float64(7), int64(3)
memory usage: 70.7 KB

INFO:file_logger:interest_rates info: None
INFO:file_logger:interest_rates NaN Count: (3196, 904)
INFO:file_logger:interest_rates head:    Year  Month  Day  Federal Funds Ta
rget Rate  Federal Funds Upper Target  \
0  1954      7    1                        NaN                         NaN
1  1954      8    1                        NaN                         NaN
2  1954      9    1                        NaN                         NaN
3  1954     10    1                        NaN                         NaN
4  1954     11    1                        NaN                         NaN

   Federal Funds Lower Target  Effective Federal Funds Rate  \
0                         NaN                          0.80
1                         NaN                          1.22
2                         NaN                          1.06
3                         NaN                          0.85
4                         NaN                          0.83

   Real GDP (Percent Change)  Unemployment Rate  Inflation Rate
0                        4.6                5.8             NaN
1                        NaN                6.0             NaN
2                        NaN                6.1             NaN
3                        8.0                5.7             NaN
4                        NaN                5.3             NaN
```

Economic Data

Look over datasets pre-scrubbing and transformation...

In [ ]:
```
logger.info(gdp_year.info())
plt.figure(figsize=(10.5,7))
sns.lineplot(x='date',y="change-current",
             data=gdp_year);

plt.title('GDP - Percent change based on current dollars')
plt.show()
```

In [ ]:
```
plt.figure(figsize=(10.5,7))
sns.lineplot(x='date',y="level-current",
             data=gdp_year);

plt.title('GDP - GDP in billions of current dollars')
plt.show()
```

In [ ]:
```
gdp_year.head()
```

In [ ]:
```
gdp_quarter.head()
```

In [ ]:
```
# interest rate -  keep Year, Month, Federal Funds Target Rate - get rid of t
ir = interest_rates_dh[~interest_rates_dh['Federal Funds Target Rate'].isna()
logger.info(ir.Year.unique())
ir.head()
```

In [ ]:
```
# inflation_consumer - filter on Country = 'United States', keep Year, Inflat
ic = inflation_consumer[inflation_consumer.Country.str.contains('United State
ic.head()
```

In [ ]:
```
# education_budget - keep Year, Value
education_budget.head()
```

In [ ]:
```
# population - keep Year, Value - drop the rest
pop = population[population['Country Name'].str.contains('United States')]
pop.head()
```

In [ ]:
```
investor_flow_monthly.head()
```

In [ ]:
```
housing_price_cities.head()
```

In [ ]:
```
# household_income - keep Year, Number(thousands), Top 5 percent
household_income.head()
```

In [ ]:
```
# employment - interesting attributes year, population, labor_force, employed
employment.head()
```

In [ ]:
```
cpi.head()
```

In [ ]:
```
cash_surp_def.head()
```

In [ ]:
```
bonds_yeilds_10y.head()
```

SCRUB / CLEAN

Clean and perform initial transformations steps of the data

REAL ESTATE DATATSETS - ZILLOW

In [16]:
```
# REAL ESTATE DATA
# Region Name is the zip code - rename for clarity
for k,v in re_datasets.items():
    v = v.rename(index=str, columns={'RegionName':'ZipCode'})
    v.ZipCode = v.ZipCode.astype(str)
    re_datasets[k] = v
```

In [17]:
```
# REAL ESTATE DATA
# convert ZipCode field to strings
keep_year_month_cols = []
month = 1
for y in ts_col_years:
    m = ''
    for i in range(1,13):
        if i < 10:
            m = '0'+str(month)
        else:
            m = str(month)
        month = month+1

        keep_year_month_cols.append(f'{y}-{m}')

    month = 1
#keep_year_month_cols
```

In [18]:

```
# REAL ESTATE DATA
# remove certain columns
# keep years
#ts_col_years
# un needed columns
dropCols = ['RegionID','SizeRank','City','Metro','CountyName']
# remove columns dates prior to 1997
pre1997Cols = ['1996-04','1996-05','1996-06','1996-07','1996-08','1996-09','1
post2018Cols = ['2019-01','2019-02', '2019-03', '2019-04', '2019-05', '2019-0

for k,v in re_datasets.items():
    # drop category columns that aren't useful
    for c in dropCols:
        if c in v.columns:
            v = v.drop(columns=c)
    # drop columns pre 1997
    for c in pre1997Cols:
        if c in v.columns:
            v = v.drop(columns=c)
    # drop columns post 2018
    for c in post2018Cols:
        if c in v.columns:
            v = v.drop(columns=c)

    # filter out by selected focus state ('WA')
    v = v[v.State==focus_state]

    re_datasets[k] = v
```

In [19]:

```
# fill NaN with median value
for k,v in re_datasets.items():
    a = v[['ZipCode','State']]
    b = v.drop(columns=['ZipCode','State'])
    b = b.T
    for c in b:
        median = np.median(b[c].sort_values().dropna())
        b[c].fillna(median,inplace=True)
    b = b.T
    re_datasets[k] = pd.concat([a,b],axis=1)
```

In [20]:
```
# REAL ESTATE DATA
# create a set of training datasets
re_datasets_train = {}
re_datasets_validate = {}

for k,v in re_datasets.items():
    # drop 2018
    i = 0
    df = pd.DataFrame(v[['ZipCode','State']])
    for c in v.columns:
        if ts_validate_year in c:

            if i == 0:
                df = v[c]
            else:
                df2 = v[c]
                df = pd.concat([df,df2], axis=1)

            v = v.drop(columns=c)

        i=i+1

    re_datasets_validate[k] = df
    re_datasets_train[k] = v
```

In [ ]:
```
sfr = re_datasets_validate['Single_Family_Residence']
sfr.head()
```

INTEREST RATE DATASET - KAGGEL

In [ ]:
```
# INTEREST RATE DATASET
# Rename column names - easier to work with...
logger.info(f'interest_rates.columns before renaming... \n{list(interest_rate
interest_rates = interest_rates.rename(index=str, columns={'Federal Funds Tar
                                                            'Federal Funds Uppe
                                                            'Federal Funds Lowe
                                                            'Effective Federal 
                                                            'Real GDP (Percent 
                                                            'Unemployment Rate'
                                                            'Inflation Rate':'I
logger.info(f'interest_rates.columns after renaming... \n{list(interest_rates
```

In [ ]:
```
# INTEREST RATE DATASET
rt.plot_corr_heatmap(interest_rates,interest_rates.drop(columns=['Year', 'Mon
```

In [ ]:
```
# look at distributions of dataset elements, determin best methods for cleani
cols =['Inflation_Rate','FF_Target_Rate', 'FF_Upper_Target', 'FF_Lower_Target
sns.boxplot(data=interest_rates[cols], orient='h',palette='Set2');
sns.swarmplot(data=interest_rates[cols], orient='h',palette='Set2');
```

In [ ]:
```
interest_rates.tail(50)
```

ECONOMIC DATASETS - DATAHUB.IO

In [ ]:
```
# interest_rates
# interest rate -  keep Year, Month, Federal Funds Target Rate - get rid of t
ir = interest_rates_dh[~interest_rates_dh['Federal Funds Target Rate'].isna()
ir = ir[['Year','Month','Federal Funds Target Rate']]
ir = ir.rename(index=str, columns={'Federal Funds Target Rate':'FF_Target_Rat
logger.debug(ir.Year.unique())
ir.head()
```

In [ ]:
```
# interest_rates - average by year
ir_y = pd.DataFrame(ir.groupby('Year').mean()['FF_Target_Rate'])
ir_y = ir_y.rename(index=str, columns={'FF_Target_Rate':'FF_Target_Rate_Avg'}
ir_y = ir_y.reset_index()
ir_y.head()
```

In [ ]:
```
# inflation_consumer - filter on Country = 'United States', keep Year, Inflat
ic = inflation_consumer[inflation_consumer.Country.str.contains('United State
ic = ic[['Year','Inflation']]
ic.head()
```

In [ ]:
```
# gdp_year
'''
level-current -> GDP in billions of current dollars
change-current -> GDP percent change based on current dollars
'''
gdp_y = gdp_year[['date','level-current','change-current']]
gdp_y = gdp_y.rename(index=str, columns={'date':'Year', 'level-current':'GDP'
gdp_y.head()
```

In [ ]:
```
# gdp_quarter
gdp_q = gdp_quarter[['date','level-current','change-current']]
gdp_q = gdp_q.rename(index=str, columns={'date':'Date', 'level-current':'GDP'
gdp_q['Date'] = pd.to_datetime(gdp_q['Date'])
gdp_q['Year'], gdp_q['Month'] = gdp_q['Date'].dt.year, gdp_q['Date'].dt.montI
gdp_q = gdp_q.drop(columns=['Date'])
gdp_q = gdp_q[['Year','Month','GDP','GDP_Percent_Change']]
gdp_q.head()
```

In [ ]:
```
# create a gdp_monthly by averaging the quarterly for the year

# TODO - need to group by year, then quarter, take average and span that over

gdp_m = pd.DataFrame(gdp_q.groupby('Year', as_index=False)['GDP','GDP_Percent
gdp_m = gdp_m.rename(index=str, columns={'GDP':'GDP_Avg', 'GDP_Percent_Change
gdp_m.head()
```

In [ ]:

```
# education_budget
'''
United States of America education budget analysis
United States of America Education budget to GDP analysis Data Data comes fro
BUDGET_ON_EDUCATION -> budget in millions of dollars
GDP -> GDP in millions of dollars
RATIO -> education expendature / GDP in percentage
'''
logger.debug(f'education budget, before... \n{education_budget.head()}')
eb = education_budget[['YEAR','BUDGET_ON_EDUCATION']]
eb = eb.rename(index=str, columns={'YEAR':'Year', 'BUDGET_ON_EDUCATION':'Educ
logger.debug(f'education budget, after... \n{eb.head()}')

eb.head()
```

In [ ]:

```
# population
'''
Population figures for countries, regions (e.g. Asia) and the world.
'''
# population - keep Year, Value - drop the rest
pop = population[population['Country Name'].str.contains('United States')]
pop = pop[['Year','Value']]
pop = pop.rename(index=str, columns={'Value':'Population'})
pop.head()
```

In [ ]:

```
# investor_flow_monthly
'''
Monthly net new cash flow by US investors into various mutual fund investment
'''
logger.debug(f'us investor flow monthly ... {investor_flow_monthly.head()}')

ifm_t = investor_flow_monthly[['Date','Total']]
ifm_t = ifm_t.rename(index=str, columns={'Total':'Investor_Flow'})
ifm_t['Date'] = pd.to_datetime(ifm_t['Date'])
ifm_t['Year'], ifm_t['Month'] = ifm_t['Date'].dt.year, ifm_t['Date'].dt.month

ifm_t = ifm_t[['Year','Month','Investor_Flow']]
logger.debug(f'us investor flow monthly total ... {investor_flow_monthly.head
ifm_t.head()
```

In [ ]:

```
# investor_flow_monthly - average out to yea
ifm_t_y = pd.DataFrame(ifm_t.groupby('Year').mean()['Investor_Flow'])
ifm_t_y = ifm_t_y.rename(index=str, columns={'Investor_Flow':'Investor_Flow_A
ifm_t_y = ifm_t_y.reset_index()
ifm_t_y.head()
```

In [ ]:
```
# housing_price_city
'''
US House Price Index (Case-Shiller) - narrow down to national index
'''
logger.debug(f'US House Price Index ... {housing_price_cities.head()}')

hp_index_m = housing_price_cities[['Date','National-US']]
hp_index_m = hp_index_m.rename(index=str, columns={'National-US':'National_Ho
hp_index_m['Date'] = pd.to_datetime(hp_index_m['Date'])
hp_index_m['Year'], hp_index_m['Month'] = hp_index_m['Date'].dt.year, hp_inde
hp_index_m = hp_index_m[['Year','Month','National_House_Price_Index']]

hp_index_m.head()
```

In [ ]:
```
# housing_price_city - aggregate to yearly average price index
hp_idex_y = pd.DataFrame(hp_index_m.groupby('Year').mean()['National_House_Pr
hp_idex_y = hp_idex_y.rename(index=str, columns={'National_House_Price_Index'
hp_idex_y = hp_idex_y.reset_index()
hp_idex_y.head()
```

In [ ]:
```
# household_income - keep Year, Number(thousands), Top 5 percent
'''


'''
logger.debug(f'{household_income.head()}')
hh_i = household_income[['Year','Number (thousands)']]
hh_i = hh_i.rename(index=str, columns={'Number (thousands)':'House_Hold_Incom
hh_i = hh_i.sort_values('Year')
hh_i.head()
```

In [ ]:
```
# employment
'''
US Employment and Unemployment rates since 1940. Official title:
*Employment status of the civilian noninstitutional population, 1940 to date*
'''
logger.debug(f'employment ... {employment.head()}')
emp = employment[['year','employed_total','employed_percent','unemployed','un
emp = emp.rename(index=str, columns={'year':'Year','employed_total':'Employed
emp.head()
```

In [ ]:
```
# cpi
'''
Consumer Price Index for All Urban Consumers (CPI-U) from U.S. Department Of
This is a monthly time series from January 1913. Values are U.S. city average
'''
logger.debug(f'cpi ... {cpi.head()}')
cpi_m = cpi[['Date','Index']]
cpi_m['Date'] = pd.to_datetime(cpi_m['Date'])
cpi_m['Year'], cpi_m['Month'] = cpi_m['Date'].dt.year, cpi_m['Date'].dt.montl
cpi_m = cpi_m.rename(index=str, columns={'Index':'CPI_Index'})
cpi_m = cpi_m[['Year','Month','CPI_Index']]
cpi_m.head()
```

In [ ]:
```
# cpi - yearly average
cpi_y = pd.DataFrame(cpi_m.groupby('Year').mean()['CPI_Index'])
cpi_y = cpi_y.rename(index=str, columns={'CPI_Index':'CPI_Index_Avg'})
cpi_y = cpi_y.reset_index()
cpi_y.head()
```

In [ ]:
```
# cash_surp_def
'''

'''
csd = cash_surp_def[cash_surp_def['Country Name'].str.contains('United States
csd = csd[['Year','Value']]
csd = csd.rename(index=str, columns={'Value':'Cash_Surp_Def'})
csd.head()
```

In [ ]:
```
# bonds_yeilds_10y
'''
10 year US Government Bond Yields (long-term interest rate)
10 year nominal yields on US government bonds from the Federal Reserve.
The 10 year government bond yield is considered a standard indicator of long-
'''
logger.debug(f'bonds yeilds 10y:\n{bonds_yeilds_10y.head()}')

by_10y_m = bonds_yeilds_10y[['Date','Rate']]
by_10y_m['Date'] = pd.to_datetime(by_10y_m['Date'])
by_10y_m['Year'], by_10y_m['Month'] = by_10y_m['Date'].dt.year, by_10y_m['Dat
by_10y_m = by_10y_m[['Year','Month','Rate']]
by_10y_m = by_10y_m.rename(index=str, columns={'Rate':'Bond_Yield_10y'})

by_10y_m.head()
```

In [ ]:
```
# bonds_yeilds_10y - averaged over the year
by_10y_y = pd.DataFrame(by_10y_m.groupby('Year').mean()['Bond_Yield_10y'])
by_10y_y = by_10y_y.rename(index=str, columns={'Bond_Yield_10y':'Bond_Yield_1
by_10y_y = by_10y_y.reset_index()
by_10y_y.head()
```

In [ ]:
```
# merge tables by year
ir_y['Year'] = ir_y['Year'].astype(str)
ic['Year'] = ic['Year'].astype(str)
gdp_y['Year'] = gdp_y['Year'].astype(str)
eb['Year'] = eb['Year'].astype(str)
pop['Year'] = pop['Year'].astype(str)
ifm_t_y['Year'] = ifm_t_y['Year'].astype(str)
hp_idex_y['Year'] = hp_idex_y['Year'].astype(str)
hh_i['Year'] = hh_i['Year'].astype(str)
emp['Year'] = emp['Year'].astype(str)
cpi_y['Year'] = cpi_y['Year'].astype(str)
csd['Year'] = csd['Year'].astype(str)
by_10y_y['Year'] = by_10y_y['Year'].astype(str)


ir['Year'] = ir['Year'].astype(str)
gdp_m['Year'] = gdp_m['Year'].astype(str)
ifm_t['Year'] = ifm_t['Year'].astype(str)
hp_index_m['Year'] = hp_index_m['Year'].astype(str)
cpi_m['Year'] = cpi_m['Year'].astype(str)
by_10y_m['Year'] = by_10y_m['Year'].astype(str)



datasets_to_merge_year = [ir_y,ic,gdp_y,eb,pop,ifm_t_y,hp_idex_y,hh_i,emp,cpi
datasets_to_merge_month = [ir,gdp_m,ifm_t,hp_index_m,cpi_m,by_10y_m]

# merge by year, first get overall range of overlapping years for each set

#ir_y.head()
```

In [ ]:
```
d = pd.merge(ir_y, ic, on='Year', how='left')
d = pd.merge(d, gdp_y, on='Year', how='left')
d = pd.merge(d, eb, on='Year', how='left')
d = pd.merge(d, pop, on='Year', how='left')
d = pd.merge(d, ifm_t_y, on='Year', how='left')
d = pd.merge(d, hp_idex_y, on='Year', how='left')
d = pd.merge(d, hh_i, on='Year', how='left')
d = pd.merge(d, emp, on='Year', how='left')
d = pd.merge(d, cpi_y, on='Year', how='left')
d = pd.merge(d, csd, on='Year', how='left')
d = pd.merge(d, by_10y_y, on='Year', how='left')
d.head()
```

In [ ]:
```
#d.head()
#d = d.drop(columns=['CPI_Index_Avg_x'])
#d = d.rename(index=str, columns={'CPI_Index_Avg_y':'CPI_Index_Avg'})
#d.head()
economicDf_year = d
del d
```

In [ ]:
```
ir = ir.sort_values(['Year','Month'])
#gdp_m = gdp_m.sort_values(['Year','Month'])
ifm_t = ifm_t.sort_values(['Year','Month'])
hp_index_m = hp_index_m.sort_values(['Year','Month'])
cpi_m = cpi_m.sort_values(['Year','Month'])
by_10y_m = by_10y_m.sort_values(['Year','Month'])
```

In [ ]:
```
#d = pd.merge(ir, gdp_m, on='Year', how='left')
d = pd.merge(ir, ifm_t, on='Year', how='left')
d = pd.merge(d, hp_index_m, on='Year', how='left')
d = pd.merge(d, cpi_m, on='Year', how='left')
d = pd.merge(d, by_10y_m, on='Year', how='left')
#d
```

In [ ]:
```
# cleanup memory
del interest_rates_dh
```

In [ ]:
```
economicDf_year.head()
economicDf_year.shape
```

In [ ]:
```
economicDf_year
```

In [ ]:
```
rt.plot_corr_heatmap(economicDf_year,economicDf_year.drop(columns=['Year']).c
```

In [ ]:
```
p = economicDf_year
p['National_HPI_Avg'] = np.log(p['National_HPI_Avg'])
plt.figure(figsize=(16,6))
ax = sns.lineplot(x='Year',y='National_HPI_Avg',data=p)
plt.title('National House Price Index over Time')
plt.ylabel('Log National House Price Index')

plt.show()
del p
```

In [ ]:
```
p = economicDf_year
p['FF_Target_Rate_Avg'] = np.log(p['FF_Target_Rate_Avg'])
plt.figure(figsize=(16,6))
ax = sns.lineplot(x='Year',y='FF_Target_Rate_Avg',data=p)
plt.title('Federal Target Interest over Time')
plt.ylabel('Log Interest Rate Avg')

plt.show()
del p
```

In [ ]:
```
factors = ['Year','FF_Target_Rate_Avg','GDP','GDP_Percent_Change','Inflation'
melt = pd.melt(economicDf_year[factors], ['Year'])
melt['value'] = np.log(melt['value'])
plt.figure(figsize=(16,6))
sns.lineplot(x='Year',y='value',hue='variable',
             data=melt)


plt.title('Federal Target Interest over Time')
plt.ylabel('Interest Rate Avg')
plt.show()
del melt
```

In [ ]:
```
plt.figure(figsize=(16,6))
sns.lmplot(x='FF_Target_Rate_Avg',y='National_HPI_Avg',data=economicDf_year)
plt.title('National Housing Price Value\nwith Interest Rates')
plt.xlabel('Federal Target Interest Rate')
plt.ylabel('National House Price Index Avg')
plt.show();
```

In [ ]:
```
plt.figure(figsize=(16,6))
sns.lmplot(x='Inflation',y='FF_Target_Rate_Avg',data=economicDf_year)
plt.title('Inflation compared to Interest Rates')
plt.xlabel('Inflation')
plt.ylabel('Federal Interest Rate')
plt.show();
```

2. Time Series Analysis

Time series analysis on real estate median average price by zipcode

- Single Family Home Value
- Rental Price psf
- Listing Price

Description: ...

2.1 Analysis

Transform Data

Transform Real Estate data for time series analysis

In [21]:
```
# Transform Datasets for Prophet Timeseries Analysis
# training datasets
re_datasets_train_prophet = {}
for k,v in re_datasets_train.items():
    re_datasets_train_prophet[k] = brs.dfTransformForProphet(v,['State'],'Zip

# validation datasets
re_datasets_validate_prophet = {}
for k,v in re_datasets_validate.items():
    re_datasets_validate_prophet[k] = brs.dfTransformForProphet(v,['State'],'

# full datasets
re_datasets_full_prophet = {}
for k,v in re_datasets.items():
    re_datasets_full_prophet[k] = brs.dfTransformForProphet(v,['State'],'ZipC
```

In [ ]:

2.2 Exploration

In [22]:
```
# have a look over the datasets shape after transformation
for k,v in re_datasets_train_prophet.items():
    logger.info(f'{k} shape: {v.shape}')

for k,v in re_datasets_validate_prophet.items():
    logger.info(f'{k} shape: {v.shape}')

for k,v in re_datasets_full_prophet.items():
    logger.info(f'{k} shape: {v.shape}')
```

```
INFO:file_logger:Single_Family_Residence shape: (252, 351)
INFO:file_logger:All_Homes shape: (252, 353)
INFO:file_logger:RentalPrice_PSF shape: (95, 66)
INFO:file_logger:RentalPrice_All_Homes shape: (95, 82)
INFO:file_logger:ListingPrice_All_Homes shape: (96, 261)
INFO:file_logger:Single_Family_Residence shape: (12, 351)
INFO:file_logger:All_Homes shape: (12, 353)
INFO:file_logger:RentalPrice_PSF shape: (12, 66)
INFO:file_logger:RentalPrice_All_Homes shape: (12, 82)
INFO:file_logger:ListingPrice_All_Homes shape: (12, 261)
INFO:file_logger:Single_Family_Residence shape: (264, 351)
INFO:file_logger:All_Homes shape: (264, 353)
INFO:file_logger:RentalPrice_PSF shape: (107, 66)
INFO:file_logger:RentalPrice_All_Homes shape: (107, 82)
INFO:file_logger:ListingPrice_All_Homes shape: (108, 261)
```

2.3 Model

In [ ]:
```
# perform ZipCod model creation and validation techniques
# build time series models
# perform exploratory data analysis techiques
# Build prophet timeseries models for the metro area, save to dictionary obje
'''

zipCodeModels = {}
t = 0.0
trainDir = 'train'

#make directories
for k in datasets.keys():
    if not os.path.exists(f'{modelDir}/{trainDir}/{k}'):
        os.makedirs(f'{modelDir}/{trainDir}/{k}')


with rt.elapsed_timer() as elapsed:

    for k,v in datasets_train_prophet.items():
        logger.info(f'Starting... {k} prophet modeling... elapsed time: {elap

        for zipcode, price in tqdm(v.items()):
            logger.info(f'Starting... {zipcode} prophet modeling... elapsed t
            model = brs.beProphet(zipcode,price,f'{modelDir}/{trainDir}/{k}/{

    logger.info(f'total elapsed time: {elapsed()}')
'''
```

In [ ]:
```
#make directories
for k in datasets.keys():
    if not os.path.exists(f'{imageDir}/{trainDir}/{k}'):
        os.makedirs(f'{imageDir}/{trainDir}/{k}')
```

In [ ]:

```
# train and predict future zipcode performance
'''
zipCodeModels = {}
t = 0.0
future5Dir = 'future_5'
# time series model projection time
ts_pred_periods = 12*5

#make directories
for k in datasets.keys():
    if not os.path.exists(f'{modelDir}/{future5Dir}/{k}'):
        os.makedirs(f'{modelDir}/{future5Dir}/{k}')


with rt.elapsed_timer() as elapsed:

    for k,v in datasets_full_prophet.items():
        logger.info(f'Starting... {k} prophet modeling... elapsed time: {elap

        for zipcode, price in tqdm(v.items()):
            logger.info(f'Starting... {zipcode} prophet modeling... elapsed t
            model = brs.beProphet(zipcode,price,f'{modelDir}/{future5Dir}/{k}

    logger.info(f'total elapsed time: {elapsed()}')
'''
```

In [ ]:

```
#make directories
for k in datasets.keys():
    if not os.path.exists(f'{imageDir}/{future5Dir}/{k}'):
        os.makedirs(f'{imageDir}/{future5Dir}/{k}')
```

2.4 Results

Training Data Sets

Price Trend from 1997 through 2017 - With a 12 month future prediction...

In [ ]:
```
# pull in forecast samples
ran_zips = np.random.choice(re_datasets['Single_Family_Residence'].ZipCode,1(
zipcode_eval = [] #
i = 0
m_fit = None
m_forecast = None

for data_key in re_datasets.keys():

    for z in ran_zips:
        try:
            fitFile = f'{modelDir}/{trainDir}/{data_key}/{z}_fit'
            forecastFile = f'{modelDir}/{trainDir}/{data_key}/{z}_forecast'

            with open(fitFile,'rb') as f:
                m_fit = pickle.load(f)
                logger.info(f'saved pickled timeseries model [{fitFile}] data

            with open(forecastFile,'rb') as f:
                m_forecast = pickle.load(f)
                logger.info(f'saved pickled timeseries model [{forecastFile}]


            brs.plotFit(z,m_fit,m_forecast,f'{z} Random {str(i+1)} Subset Sam
            i = i+1
            #if i >= 5: break

        except FileNotFoundError:
            logger.info('file not found...')
```

Future Prediction Trends

Price Trend from 1997 through 2018 - With a 5 year future prediction...

In [ ]:
```
# pull in forecast samples
ran_zips = np.random.choice(re_datasets['Single_Family_Residence'].ZipCode,1
zipcode_eval = [] #
i = 0
m_fit = None
m_forecast = None

for data_key in re_datasets.keys():

    for z in ran_zips:
        try:
            fitFile = f'{modelDir}/{future5Dir}/{data_key}/{z}_fit'
            forecastFile = f'{modelDir}/{future5Dir}/{data_key}/{z}_forecast'

            with open(fitFile,'rb') as f:
                m_fit = pickle.load(f)
                logger.info(f'saved pickled timeseries model [{fitFile}] data

            with open(forecastFile,'rb') as f:
                m_forecast = pickle.load(f)
                logger.info(f'saved pickled timeseries model [{forecastFile}]


            brs.plotFit(z,m_fit,m_forecast,f'{z} Random {str(i+1)} Subset Sam
            i = i+1
            #Sif i >= 5: break

        except FileNotFoundError:
            logger.info('file not found...')
```

In [ ]:
```
from fbprophet.diagnostics import cross_validation, performance_metrics
df_cv = cross_validation(m_fit, horizon='90 days')

df_p = performance_metrics(df_cv)
#df_p.head(5)
```

In [ ]:
```
from fbprophet.plot import plot_cross_validation_metric
fig = plot_cross_validation_metric(df_cv, metric='mape')
```

2.5 Interpret

3. Clustering

- K-means - unsupervised
- Mean-Shift - unsupervised

Description: Run k-means for three choices for k and choose the best.

3.1 K-means Clustering

Python package: scikit-learn v0.21.3 [sklearn.cluster.KMeans (https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans)](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans)

Description: ...

3.1.1 Analysis

In [37]:

```
# which are the best forecasters - group into 4 classes

#from fbprophet.diagnostics import cross_validation, performance_metrics
#df_cv = cross_validation()
# focus on single familey homes
zipcodes = re_datasets['Single_Family_Residence'].ZipCode
data_key = 'Single_Family_Residence'
zip_ts_forcasts = {}

# get zip forcasts
for z in zipcodes:
    forecastFile = f'{modelDir}/{future5Dir}/{data_key}/{z}_forecast'

    with open(forecastFile,'rb') as f:
            m_forcast = pickle.load(f)
            zip_ts_forcasts[z] = m_forcast
            logger.info(f'saved pickled timeseries model [{forecastFile}] dat


#m_forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']]
```

```
_Family_Residence/98925_fit] dataset found...
INFO:file_logger:saved pickled timeseries model [./models/future_5/Single
_Family_Residence/98925_fit] dataset found...
INFO:file_logger:saved pickled timeseries model [./models/future_5/Single
_Family_Residence/98925_fit] dataset found...
INFO:file_logger:saved pickled timeseries model [./models/future_5/Single
_Family_Residence/98925_fit] dataset found...
INFO:file_logger:saved pickled timeseries model [./models/future_5/Single
_Family_Residence/98925_fit] dataset found...
INFO:file_logger:saved pickled timeseries model [./models/future_5/Single
_Family_Residence/98925_fit] dataset found...
INFO:file_logger:saved pickled timeseries model [./models/future_5/Single
_Family_Residence/98925_fit] dataset found...
INFO:file_logger:saved pickled timeseries model [./models/future_5/Single
_Family_Residence/98925_fit] dataset found...
INFO:file_logger:saved pickled timeseries model [./models/future_5/Single
_Family_Residence/98925_fit] dataset found...
INFO:file_logger:saved pickled timeseries model [./models/future_5/Single
_Family_Residence/98925_fit] dataset found...
INFO:file_logger:saved pickled timeseries model [./models/future_5/Single
_Family_Residence/98925_fit] dataset found...
```

In [38]:

```
98052
             ds      trend  yhat_lower  yhat_upper  trend_lower  trend_upper
\
319 2023-07-31  14.332032   13.409492   15.102551    13.423305    15.095156
320 2023-08-31  14.342047   13.414233   15.142978    13.402184    15.121482
321 2023-09-30  14.351739   13.395264   15.168953    13.382132    15.155094
322 2023-10-31  14.361753   13.372179   15.208130    13.361862    15.189825
323 2023-11-30  14.371445   13.345755   15.215508    13.342156    15.223437

     additive_terms  additive_terms_lower  additive_terms_upper    yearly
\
319        0.003592              0.003592              0.003592  0.003592
320        0.007343              0.007343              0.007343  0.007343
321        0.003960              0.003960              0.003960  0.003960
322       -0.001307             -0.001307             -0.001307 -0.001307
323       -0.005750             -0.005750             -0.005750 -0.005750

     yearly_lower  yearly_upper  multiplicative_terms  \
319      0.003592      0.003592                   0.0
320      0.007343      0.007343                   0.0
321      0.003960      0.003960                   0.0
322     -0.001307     -0.001307                   0.0
323     -0.005750     -0.005750                   0.0

     multiplicative_terms_lower  multiplicative_terms_upper       yhat
319                         0.0                         0.0  14.335625
320                         0.0                         0.0  14.349390
321                         0.0                         0.0  14.355698
322                         0.0                         0.0  14.360447
323                         0.0                         0.0  14.365695
```

3.1.2 Exploration

In [110]:

```
# get all of the zip code forecast predictions and prep for kmea

zip_forecasts = None
i = 0
for k,v in zip_ts_forcasts.items(): #351
    p = zip_ts_forcasts[k]
    p = p.drop(columns=['ds']) #drop the date field, adds no value for cluste
    p['ZipCode'] = z
    if i == 0:
        zip_forecasts = p
    else:
        zip_forecasts = pd.concat([zip_forecasts,p])
    i=i+1

rt.save_df(zip_forecasts, f'{dataDir}/zip_forecasts.pkl')
```

In [105]:

```
zip_forecasts.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 113724 entries, 0 to 323
Data columns (total 16 columns):
trend                          113724 non-null float64
yhat_lower                     113724 non-null float64
yhat_upper                     113724 non-null float64
trend_lower                    113724 non-null float64
trend_upper                    113724 non-null float64
additive_terms                 113724 non-null float64
additive_terms_lower           113724 non-null float64
additive_terms_upper           113724 non-null float64
yearly                         113724 non-null float64
yearly_lower                   113724 non-null float64
yearly_upper                   113724 non-null float64
multiplicative_terms           113724 non-null float64
multiplicative_terms_lower     113724 non-null float64
multiplicative_terms_upper     113724 non-null float64
yhat                           113724 non-null float64
ZipCode                        113724 non-null object
dtypes: float64(15), object(1)
memory usage: 14.7+ MB
```

3.1.3 Model

In [106]:

```
from sklearn.cluster import KMeans, SpectralClustering
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_samples, silhouette_score

def build_kmeans(n_clusters,random_state,n_jobs):
    km = KMeans(
        n_clusters=8,
        init="k-means++",
        n_init=10,
        max_iter=300,
        tol=0.0001,
        precompute_distances="auto",
        verbose=0,
        random_state=None,
        copy_x=True,
        n_jobs=None,
        algorithm="auto")

    return km
```

In [118]:

```
%%time
# build a kmeans clustering model
# cluster at 3 classes that will represent buy - rent - sell
# standardize the data
#X_std = fit_transform(zip_forecasts)

sse = {}
for k in range(1,10):
    km = build_kmeans(n_clusters=k,random_state=42,n_jobs=None)
    X_std = km.fit_transform(zip_forecasts)
    kmeans = km.fit(X_std)
    zip_forecasts['label'] = kmeans.labels_
    sse[k] = kmeans.inertia_ # Inertia: sum of distances of samples to their
plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel('Number of clusters')
plt.ylabel('SSE')
plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<timed exec> in <module>

AttributeError: 'numpy.ndarray' object has no attribute 'labels_'
```

In [ ]:

```
%%time
sse = {}
for k in range(2,6):

    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    km = build_kmeans(n_clusters=k,random_state=42,n_jobs=None)
    X_std = km.fit_transform(zip_forecasts)
    labels = km.fit_predict(X_std)
    centroids = km.cluster_centers_

    # Get silhouette samples
    silhouette_vals = silhouette_samples(X_std, labels)

    # Silhouette plot
    y_ticks = []
    y_lower, y_upper = 0, 0
    for i, cluster in enumerate(np.unique(labels)):
        cluster_silhouette_vals = silhouette_vals[labels == cluster]
        cluster_silhouette_vals.sort()
        y_upper += len(cluster_silhouette_vals)
        ax1.barh(range(y_lower, y_upper), cluster_silhouette_vals, edgecolor=
        ax1.text(-0.03, (y_lower + y_upper) / 2, str(i + 1))
        y_lower += len(cluster_silhouette_vals)

    # Get the average silhouette score and plot it
    avg_score = np.mean(silhouette_vals)
    ax1.axvline(avg_score, linestyle='--', linewidth=2, color='green')
    ax1.set_yticks([])
    ax1.set_xlim([-0.1, 1])
    ax1.set_xlabel('Silhouette coefficient values')
    ax1.set_ylabel('Cluster labels')
    ax1.set_title('Silhouette plot for the various clusters', y=1.02);

    # Scatter plot of data colored with labels
    ax2.scatter(X_std[:, 0], X_std[:, 1], c=labels)
    ax2.scatter(centroids[:, 0], centroids[:, 1], marker='*', c='r', s=250)
    ax2.set_xlim([-2, 2])
    ax2.set_xlim([-2, 2])
    ax2.set_xlabel('Eruption time in mins')
    ax2.set_ylabel('Waiting time to next eruption')
    ax2.set_title('Visualization of clustered data', y=1.02)
    ax2.set_aspect('equal')
    plt.tight_layout()
    plt.suptitle(f'Silhouette analysis using k = {k}',
                 fontsize=16, fontweight='semibold', y=1.05);
```

In [115]:

```
logger.info(f'kmeans class labels... {kmeans.labels_}')
logger.info(f'kmeans parameters... {kmeans.get_params()}')
logger.info(f'{zip_forecasts["label"].unique()}')
kmeans.cluster_centers_[kmeans.labels_]
```

```
INFO:file_logger:kmeans class labels... [1 2 1 ... 7 7 7]
INFO:file_logger:kmeans parameters... {'algorithm': 'auto', 'copy_x': True,
'init': 'k-means++', 'max_iter': 300, 'n_clusters': 8, 'n_init': 10, 'n_job
s': None, 'precompute_distances': 'auto', 'random_state': None, 'tol': 0.00
01, 'verbose': 0}
INFO:file_logger:[1 2 3 5 4 6 7 0]
```

Out[115]:

```
array([[-0.16592725, -0.0576433 , -0.23247871, ...,  0.        ,
        -0.16555623,  0.        ],
       [-0.97768016, -0.94461094, -0.898865  , ...,  0.        ,
        -0.97401381,  0.        ],
       [-0.16592725, -0.0576433 , -0.23247871, ...,  0.        ,
        -0.16555623,  0.        ],
       ...,
       [ 0.45248853, -0.06616244,  0.83422806, ...,  0.        ,
         0.46847868,  0.        ],
       [ 0.45248853, -0.06616244,  0.83422806, ...,  0.        ,
         0.46847868,  0.        ],
       [ 0.45248853, -0.06616244,  0.83422806, ...,  0.        ,
         0.46847868,  0.        ]])
```

In [113]:

```
km_centroids = kmeans.centroids
# Plot the clustered data
fig, ax = plt.subplots(figsize=(6, 6))
plt.scatter(zip_forecasts[kmeans.labels == 0, 0], zip_forecasts[kmeans.labels
            c='green', label='cluster 1')
plt.scatter(zip_forecasts[kmeans.labels == 1, 0], zip_forecasts[kmeans.labels
            c='blue', label='cluster 2')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=300,
            c='r', label='centroid')
plt.legend()
plt.xlim([-2, 2])
plt.ylim([-2, 2])
plt.xlabel('')
plt.ylabel('')
plt.title('Visualization of clustered data', fontweight='bold')
ax.set_aspect('equal');
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-113-b5a26286db74> in <module>
----> 1 km_centroids = kmeans.centroids
      2 # Plot the clustered data
      3 fig, ax = plt.subplots(figsize=(6, 6))
      4 plt.scatter(zip_forecasts[kmeans.labels == 0, 0], zip_forecasts[kme
ans.labels == 0, 1],
      5             c='green', label='cluster 1')

AttributeError: 'KMeans' object has no attribute 'centroids'
```

3.1.4 Results

In [ ]:

In [ ]:

In [ ]:

In [ ]:

3.1.5 Interpret

In [ ]:

3.2 Mean Shift Clustering

Python package: scikit-learn v0.21.3 [sklearn.cluster.MeanShift (https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MeanShift.html#sklearn.cluster.MeanShift)](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MeanShift.html#sklearn.cluster.MeanShift)

Description: ...

3.2.1 Analysis

In [ ]:

3.2.2 Exploration

In [ ]:

3.2.3 Model

In [ ]:

```
def build_meanShift(seeds,cluster_all,n_jobs):
    ms = MeanShift(
            bandwidth=None,
            seeds=None,
            bin_seeding=False,
            min_bin_freq=1,
            cluster_all=True,
            n_jobs=None)

    return ms
```

In [ ]:

In [ ]:

In [ ]:

3.2.4 Results

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

3.2.5 Interpret

In [ ]:

In [ ]:

In [ ]:

4. Decision Tree

- Decision Tree - supervised
  - Include three different trees and their visualizations

Python package: [scikit-learn sklearn.tree.DecisionTreeClassifier (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)

Build a decision tree model. Tune the parameters, such as the pruning options, and report the 3-fold CV accuracy.

4.1 Analysis

In [ ]:

4.2 Exploration

In [ ]:

4.3 Model

In [ ]:

```
def build_tree(random_state,max_depth,min_samples_split):
    tree = DecisionTreeClassifier(
        criterion="gini",
        splitter="best",
        max_depth=None,
        min_samples_split=2,
        min_samples_leaf=1,
        min_weight_fraction_leaf=0.0,
        max_features=None,
        random_state=None,
        max_leaf_nodes=None,
        min_impurity_decrease=0.0,
        min_impurity_split=None,
        class_weight=None,
        presort=False)

    return tree
```

In [ ]:

In [ ]:

In [ ]:

4.4 Results

In [ ]:

In [ ]:

In [ ]:

4.5 Interpret

In [ ]:

5. Naive Bayes

Python Package: SciKit-Learn [Gaussian Naive Bayes (https://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes)](https://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes)

Build a naïve Bayes model. Tune the parameters, such as the discretization options, to compare results.

5.1 Analysis

In [ ]:

5.2 Exploration

In [ ]:

5.3 Model

In [ ]:

```
def build_nb(priors):
    nb = GaussianNB(priors=None, var_smoothing=1e-09)

    return nb
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

5.4 Results

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

5.5 Interpret

In [ ]:

In [ ]:

6. Support Vector Classification - SVMs

Python Package: scikit-learn v0.21.3 [sklearn.svm.SVC (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC)](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC)

6.1 Analysis

In [ ]:

6.2 Exploration

In [ ]:

6.3 Model

In [ ]:

```python
def build_svm(kernel,verbose=True):
    # base SVC model
    svc_base = SVC(C=1.0,                         # Penalty parameter C of th
                   cache_size=200,                # Specify the size of the k
                   class_weight=None,             # Set the parameter C of cl
                   coef0=0.0,                     # Independent term in kerne
                   decision_function_shape='ovr', # Whether to return a one-v
                   degree=3,                      # Degree of the polynomial
                   gamma='auto',                  # Kernel coefficient for 'r
                   kernel=kernel,                 # Specifies the kernel type
                   max_iter=-1,                   # Hard limit on iterations
                   probability=False,             # Whether to enable probabi
                   random_state=None,             # The seed of the pseudo ra
                   shrinking=True,                # Whether to use the shrink
                   tol=0.001,                     # Tolerance for stopping cr
                   verbose=verbose                # Enable verbose output. No
                  )
    return svc_base
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

6.4 Results

In [ ]:

In [ ]:

In [ ]:

In [ ]:

6.5 Interpret

In [ ]:

7. Association Rule Mining

- Unsupervised
  Description: Offer the top 10 rules for the highest sup, the top 10 for conf, and the top 10 for lift. All rules must have at least one element on the left and one on the right. Also choose to set the left as a given value and show the top 10 (based on the dataset and determinations.)

7.1 Analysis

In [ ]:

7.2 Exploration

In [ ]:

7.3 Model

In [ ]:

7.4 Results

In [ ]:

7.5 Interpret

In [ ]:

```

```

2. Support Vector Classification - SVMs

Python Package: scikit-learn v0.21.3 [sklearn.svm.SVC (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC)](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC)

Modeling & Evaluation Functions

Python package [sklearn.metrics.confusion_matrix (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html)](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html)

2.1 Model - Support Vector Classification

Python package:

2.1.1 Create Base SVC model

In [ ]:

```
# base SVC model
svc_base = SVC(C=1.0,                           # Penalty parameter C of the er
               cache_size=200,                  # Specify the size of the kerne
               class_weight=None,               # Set the parameter C of class
               coef0=0.0,                       # Independent term in kernel fu
               decision_function_shape='ovr',   # Whether to return a one-vs-re
               degree=3,                        # Degree of the polynomial kern
               gamma='auto',                    # Kernel coefficient for ‘rbf’,
               kernel='rbf',                    # Specifies the kernel type to
               max_iter=-1,                     # Hard limit on iterations with
               probability=False,               # Whether to enable probability
               random_state=None,               # The seed of the pseudo random
               shrinking=True,                  # Whether to use the shrinking
               tol=0.001,                       # Tolerance for stopping criter
               verbose=True                     # Enable verbose output. Note t
              )
```

In [ ]:
```
# fit the svc model
t = 0.0
with rt.elapsed_timer() as elapsed:
    #
    svc_fit = svc_base.fit(X_train, y_train)
    t = elapsed()
    if sh_logger.debug: print(f'Support Vector Classification Model Build Tim

modelsPerformance['Name'].append('svc_base')
modelsPerformance['FitTime'].append(t)

#save model to file
with open(modelBaselineDir+'svc_base','wb') as f:
    pickle.dump(svc_base,f)

with open(modelBaselineDir+'svc_fit','wb') as f:
    pickle.dump(svc_fit,f)
```

In [ ]:
```
# Score the svc model
t = 0.0
with rt.elapsed_timer() as elapsed:
    svc_score =  svc_base.score(X_val, y_val)
    t = elapsed()
    if sh_logger.info: print(f'Support Vector Classification Model Fit Score:
    if sh_logger.debug: print(f'Support Vector Classification Model Fit Score

modelsPerformance['TestAccuracyScore'].append(svc_score)
modelsPerformance['ScoreTime'].append(t)

# save score to file
with open(modelBaselineDir+'svc_score','wb') as f:
    pickle.dump(svc_score,f)
```

2.1.2 Fit Model Prediciton - SVC

In [ ]:
```
#%%time
# predictions of test set split from training set
t = 0.0
with rt.elapsed_timer() as elapsed:
    svc_pred = svc_base.predict(X_val)
    t = elapsed()
    if sh_logger.debug: print(f'Support Vector Classification Predict Time: [

modelsPerformance['PredictTime'].append(t)

# save score to file
with open(modelBaselineDir+'svc_pred','wb') as f:
    pickle.dump(svc_pred,f)
```

In [ ]:

```
%%time
if sh_logger.debug: print(f'y_val size: {y_val.size} svc_pred size: {svc_prec

#correct and inccorrect
correct = np.nonzero(svc_pred==y_val)[0]
incorrect = np.nonzero(svc_pred!=y_val)[0]

d = {'Label':y_val, 'Prediction':svc_pred}
svcPredictionsDf = pd.DataFrame(data=d)
if sh_logger.debug: print(f'Support Vector Classification DF Shape: {svcPredi

# which test observations were miss classified
svc_missClassified_DT = svcPredictionsDf[(svcPredictionsDf['Label'] != svcPre

if sh_logger.debug: print(f'Miss Classified DF Shape: {svc_missClassified_DT.
if sh_logger.debug: print(f'Miss Classified Percent: {svc_missClassified_DT.s
if sh_logger.info: print(f'Total Number of points: [{X_val.shape[0]}]  Mislab
```

In [ ]:

```
# sample plot of correctly predicted images
plt.figure(figsize=(10, 7.5))
for i, cor in enumerate(np.random.choice(correct,9,replace=False)):
    # plot subplot of incorrect predictions
    plt.subplot(3,3,i+1)
    plt.imshow(X_val[cor].reshape(28,28), cmap='gray', interpolation='none')
    plt.title(f'Predicted: {svc_pred[cor]}, Predicted Label: {class_to_label[
              \nClass: {y_val[cor]} Class Label:{class_to_label[y_val[cor]]}'
              fontsize=10)


plt.tight_layout()
plt.savefig(f'{imageDir}svc_sample_correct_images.png', dpi=300)
plt.show()
```

In [ ]:

```
# plot sample of incorrect
plt.figure(figsize=(10, 7.5))
for i, inc in enumerate(np.random.choice(incorrect,9,replace=False)):
    # plot subplot of incorrect predictions
    plt.subplot(3,3,i+1)
    plt.imshow(X_val[inc].reshape(28,28), cmap='gray', interpolation='none')
    plt.title(f'Predicted: {svc_pred[inc]}, Predicted Label: {class_to_label[
              \nClass: {y_val[inc]} Class Label:{class_to_label[y_val[inc]]}'
              fontsize=10)


plt.tight_layout()
plt.savefig(f'{imageDir}svc_sample_incorrect_images.png', dpi=300)
plt.show()
```

2.1.3 Evaluate Model

In [ ]:
```
misLabeled = (y_val != svc_pred).sum()/X_val.shape[0]
svmAccuractelyLabeled = 1-misLabeled

if sh_logger.info: print(f'Total Number of points: [{X_val.shape[0]}]  Mislab
if sh_logger.info: print(f'Percent Mislabeled: [{((y_val != svc_pred).sum()/X
if sh_logger.info: print(f'Percent Accurately Labeled: [{svmAccuractelyLabele

modelsPerformance['PredictAccuracyScore'].append(svmAccuractelyLabeled)

#print classification report table
targetNames = ["Class{}".format(i) for i in range(n_classes)]
if sh_logger.info: print(f'\n{classification_report(y_val, svc_pred, target_n

#print confusion matrix report
cm = confusion_matrix(y_val,svc_pred, labels=[0,1,2,3,4,5,6,7,8,9])
if sh_logger.info: print(f'\nSVM Base Confusion Matrix Report:\n{cm}')

# plot confusion matrix evaluation
if sh_logger.info: rt.plot_confusion_matrix(cm,classes=[0,1,2,3,4,5,6,7,8,9])
```

In [ ]:
```
sns.swarmplot(x='Label',y='Prediction',data=svc_missClassified_DT);
```

2.1.4 Tune Model Performance

For more information on tuning an SVM classifier, go [here (https://scikit-learn.org/stable/modules/svm.html#svm-classification)](https://scikit-learn.org/stable/modules/svm.html#svm-classification) for details.
Tips:

- Consider performing dimensionality reduction (PCA, ICA, or Feature selection) beforehand to give your tree a better chance of finding features that are discriminative.
- Balance your dataset before training to prevent the tree from being biased toward the classes that are dominant.
- All decision trees use np.float32 arrays internally.
- If the input matrix X is very sparse, it is recommended to convert to sparse csc_matrix before calling fit and sparse csr_matrix before calling predict. Training time can be orders of magnitude faster for a sparse matrix input compared to a dense matrix when features have zero values in most of the samples.

In [ ]:
```
# tune model
```

3. Model - RandomForestClassifier

Python Package: scikit-learn v0.21.3 [sklearn.ensemble.RandomForestClassifier (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default).

3.1 Create Models - RandomForestClassifier

Cross-Validate at 3 folds...

In [ ]:
```
# pre-process datasets
```

Random Forest Attributes

See scikit learn [glossary (https://scikit-learn.org/stable/glossary.html#term-warm-start)](https://scikit-learn.org/stable/glossary.html#term-warm-start) for indepth details.

In [ ]:
```
rf_base = RandomForestClassifier(n_estimators=100,              # The number o
                                 criterion="gini",              # The function
                                 max_depth=None,                # The maximum 
                                 min_samples_split=2,           # The minimum 
                                 min_samples_leaf=1,            # The minimum 
                                 min_weight_fraction_leaf=0.0,  # The minimum 
                                 max_features="auto",           # The number o
                                 max_leaf_nodes=None,           # Grow trees w
                                 min_impurity_decrease=0.0,     # A node will 
                                 min_impurity_split=None,       # Threshold fo
                                 bootstrap=True,                # Whether boot
                                 oob_score=False,               # Whether to u
                                 n_jobs=None,                   # The number o
                                 random_state=None,             # if int, rand
                                 verbose=2,                     # Controls the
                                 warm_start=False,              # When set to 
                                 class_weight=None              # Weights asso
                                )
```

In [ ]:
```
# fit the Random Forest model
t = 0.0
with rt.elapsed_timer() as elapsed:
    #
    rf_base_fit = rf_base.fit(X_train, y_train)
    t = elapsed()
    if sh_logger.debug: print(f'Random Forest Classification Model Build Time

modelsPerformance['Name'].append('rf_base')
modelsPerformance['FitTime'].append(t)

#save model to file
with open(modelBaselineDir+'rf_base','wb') as f:
    pickle.dump(rf_base,f)

with open(modelBaselineDir+'rf_base_fit','wb') as f:
    pickle.dump(rf_base_fit,f)
```

In [ ]:
```
rf_base.get_params(deep=True)
```

In [ ]:
```
# Score the Random Forest model
t = 0.0
with rt.elapsed_timer() as elapsed:
    rf_base_score =  rf_base.score(X_val, y_val)
    t = elapsed()
    if sh_logger.info: print(f'Random Forest Base Classification Model Fit Sc
    if sh_logger.debug: print(f'Random Forest Base Classification Model Fit S

modelsPerformance['TestAccuracyScore'].append(rf_base_score)
modelsPerformance['ScoreTime'].append(t)

# save score to file
with open(modelBaselineDir+'rf_base_score','wb') as f:
    pickle.dump(rf_base_score,f)
```

In [ ]:
```

modelsPerformance
```

3.2 Fit Model Prediction - Random Forest

In [ ]:
```
%%time
# predictions of test set split from training set
t = 0.0
with rt.elapsed_timer() as elapsed:
    rf_base_pred = rf_base.predict(X_val)
    t = elapsed()
    if sh_logger.debug: print(f'Random Forest Base Classification Predict Tim

modelsPerformance['PredictTime'].append(t)

# save score to file
with open(modelBaselineDir+'rf_base_pred','wb') as f:
    pickle.dump(rf_base_pred,f)
```

In [ ]:
```
%%time
if sh_logger.debug: print(f'y_val size: {y_val.size} rf_base_pred size: {rf_b

#correct and inccorrect
correct = np.nonzero(rf_base_pred==y_val)[0]
incorrect = np.nonzero(rf_base_pred!=y_val)[0]

d = {'Label':y_val, 'Prediction':rf_base_pred}
rf_base_PredictionsDf = pd.DataFrame(data=d)
if sh_logger.debug: print(f'Random Forest Base Classification DF Shape: {rf_b

# which test observations were miss classified
rf_base_missClassified_DT = rf_base_PredictionsDf[(rf_base_PredictionsDf['Lab

if sh_logger.debug: print(f'Miss Classified DF Shape: {rf_base_missClassified
if sh_logger.debug: print(f'Miss Classified Percent: {rf_base_missClassified_
if sh_logger.info: print(f'Total Number of points: [{X_val.shape[0]}]  Mislab
```

In [ ]:
```
# sample plot of correctly predicted images
plt.figure(figsize=(10, 7.5))
for i, cor in enumerate(np.random.choice(correct,9,replace=False)):
    # plot subplot of incorrect predictions
    plt.subplot(3,3,i+1)
    plt.imshow(X_val[cor].reshape(28,28), cmap='gray', interpolation='none')
    plt.title(f'Predicted: {rf_base_pred[cor]}, Predicted Label: {class_to_la
              \nClass: {y_val[cor]} Class Label:{class_to_label[y_val[cor]]}'
              fontsize=10)


plt.tight_layout()
plt.savefig(f'{imageDir}rf_base_sample_correct_images.png', dpi=300)
plt.show()
```

In [ ]:
```
# plot sample of incorrect
plt.figure(figsize=(10, 7.5))
for i, inc in enumerate(np.random.choice(incorrect,9,replace=False)):
    # plot subplot of incorrect predictions
    plt.subplot(3,3,i+1)
    plt.imshow(X_val[inc].reshape(28,28), cmap='gray', interpolation='none')
    plt.title(f'Predicted: {rf_base_pred[inc]}, Predicted Label: {class_to_la
              \nClass: {y_val[inc]} Class Label:{class_to_label[y_val[inc]]}'
              fontsize=10)


plt.tight_layout()
plt.savefig(f'{imageDir}rf_base_sample_incorrect_images.png', dpi=300)
plt.show()
```

3.3 Evaluate Model - Random Forest Base

In [ ]:
```
misLabeled = (y_val != rf_base_pred).sum()/X_val.shape[0]
rfBaseAccuractelyLabeled = 1-misLabeled

if sh_logger.info: print(f'Total Number of points: [{X_val.shape[0]}]  Mislab
if sh_logger.info: print(f'Percent Mislabeled: [{((y_val != rf_base_pred).sum
if sh_logger.info: print(f'Percent Accurately Labeled: [{rfBaseAccuractelyLab

modelsPerformance['PredictAccuracyScore'].append(rfBaseAccuractelyLabeled)

#print classification report table
targetNames = ["Class{}".format(i) for i in range(n_classes)]
if sh_logger.info: print(f'\n{classification_report(y_val, rf_base_pred, targ

#print confusion matrix report
cm = confusion_matrix(y_val,rf_base_pred, labels=[0,1,2,3,4,5,6,7,8,9])
if sh_logger.info: print(f'\nSVM Base Confusion Matrix Report:\n{cm}')

# plot confusion matrix evaluation
if sh_logger.info: rt.plot_confusion_matrix(cm,classes=[0,1,2,3,4,5,6,7,8,9])
```

In [ ]:
```
ax = rf_base_missClassified_DT['Label'].plot.hist(bins=10)
```

In [ ]:
```
sns.barplot(x='Label',y='Prediction',data=rf_base_missClassified_DT);
```

In [ ]:
```
sns.swarmplot(x='Label',y='Prediction',data=rf_base_missClassified_DT);
```

3.4 Tune Models

For details on how to tune a Random Forest Classifier, go [here (https://scikit-learn.org/stable/modules/ensemble.html#random-forest-parameters)](https://scikit-learn.org/stable/modules/ensemble.html#random-forest-parameters) for details.

In [ ]:
```
# perform model creation and validation techniques
```

3.5 iNterpret Models

Interpret the model results, make knowledge based recommendations

In [ ]:
```
# perform interpretation steps
```

4. K Neighbors Classifier

Python Package: scikit-learn v0.21.3 [sklearn.neighbors.KNeighborsClassifier (https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.)

4.1 Explore

Explore the datasets

In [ ]:
```
if sh_logger.debug: print(f'{data.shape} {target.shape}')
if sh_logger.debug: print(f'{data[1:10]} \n{target[1:10]}')
```

4.2 Model

Create models

4.2.1 Create Base KNeighborsClassifier Model

In [ ]:
```
modelsPerformance
```

In [ ]:
```
# initialize base KNeighborsClassifier
kNN_base = KNeighborsClassifier(n_neighbors=5,
                                weights="uniform",    # weight function used i
                                algorithm="auto",     # Algorithm used to comp
                                leaf_size=30,         # Leaf size passed to Ba
                                p=2,                  # Power parameter for th
                                metric="minkowski",   # the distance metric to
                                metric_params=None,   # Additional keyword arg
                                n_jobs=None)          # The number of parallel
```

In [ ]:
```
# fit the kNN model
t = 0.0
with rt.elapsed_timer() as elapsed:
    #
    kNN_fit = kNN_base.fit(X_train, y_train)
    t = elapsed()
    if sh_logger.debug: print(f'K-Nearest Neighbors Classification Model Buil

modelsPerformance['Name'].append('kNN_base')
modelsPerformance['FitTime'].append(t)

#save model to file
with open(modelBaselineDir+'kNN_base','wb') as f:
    pickle.dump(kNN_base,f)

with open(modelBaselineDir+'kNN_fit','wb') as f:
    pickle.dump(kNN_fit,f)
```

In [ ]:
```
# Score the kNN base model
t = 0.0
with rt.elapsed_timer() as elapsed:
    kNN_base_score =  kNN_base.score(X_val, y_val)
    t = elapsed()
    if sh_logger.info: print(f'K-Nearest Neighbors Classification Model Fit S
    if sh_logger.debug: print(f'K-Nearest Neighbors Classification Model Fit 

modelsPerformance['TestAccuracyScore'].append(kNN_base_score)
modelsPerformance['ScoreTime'].append(t)

# save score to file
with open(modelBaselineDir+'kNN_base_score','wb') as f:
    pickle.dump(kNN_base_score,f)
```

In [ ]:
```
kNN_base.get_params(deep=True)
```

In [ ]:
```

```

4.2.2 Fit Model Prediction - kNN Base

In [ ]:
```
#%%time
# predictions of test set split from training set
t = 0.0
with rt.elapsed_timer() as elapsed:
    kNN_base_pred = kNN_base.predict(X_val)
    t = elapsed()
    if sh_logger.debug: print(f'K-Nearest Neighbors Base Classification Model

modelsPerformance['PredictTime'].append(t)

# save score to file
with open(modelBaselineDir+'kNN_base_pred','wb') as f:
    pickle.dump(kNN_base_pred,f)
```

In [ ]:
```
%%time
if sh_logger.debug: print(f'y_val size: {y_val.size} svc_pred size: {kNN_base

#correct and inccorrect
correct = np.nonzero(kNN_base_pred==y_val)[0]
incorrect = np.nonzero(kNN_base_pred!=y_val)[0]

d = {'Label':y_val, 'Prediction':kNN_base_pred}
kNNPredictionsDf = pd.DataFrame(data=d)
if sh_logger.debug: print(f'K-Nearest Neighbors Classification Model DF Shape

# which test observations were miss classified
kNN_missClassified_DT = kNNPredictionsDf[(kNNPredictionsDf['Label'] != kNNPre

if sh_logger.debug: print(f'Miss Classified DF Shape: {kNN_missClassified_DT.
if sh_logger.debug: print(f'Miss Classified Percent: {kNN_missClassified_DT.s
if sh_logger.info: print(f'Total Number of points: [{X_val.shape[0]}]  Mislab
```

In [ ]:
```
# sample plot of correctly predicted images
plt.figure(figsize=(10, 7.5))
for i, cor in enumerate(np.random.choice(correct,9,replace=False)):
    # plot subplot of incorrect predictions
    plt.subplot(3,3,i+1)
    plt.imshow(X_val[cor].reshape(28,28), cmap='gray', interpolation='none')
    plt.title(f'Predicted: {kNN_base_pred[cor]}, Predicted Label: {class_to_l
              \nClass: {y_val[cor]} Class Label:{class_to_label[y_val[cor]]}'
              fontsize=10)


plt.tight_layout()
plt.savefig(f'{imageDir}kNN_base_sample_correct_images.png', dpi=300)
plt.show()
```

In [ ]:
```
# plot sample of incorrect
plt.figure(figsize=(10, 7.5))
for i, inc in enumerate(np.random.choice(incorrect,9,replace=False)):
    # plot subplot of incorrect predictions
    plt.subplot(3,3,i+1)
    plt.imshow(X_val[inc].reshape(28,28), cmap='gray', interpolation='none')
    plt.title(f'Predicted: {kNN_base_pred[inc]}, Predicted Label: {class_to_l
              \nClass: {y_val[inc]} Class Label:{class_to_label[y_val[inc]]}'
              fontsize=10)


plt.tight_layout()
plt.savefig(f'{imageDir}kNN_base_sample_incorrect_images.png', dpi=300)
plt.show()
```

4.2.3 Evaluate Models - kNN Base

In [ ]:
```
misLabeled = (y_val != kNN_base_pred).sum()/X_val.shape[0]
kNNAccuractelyLabeled = 1-misLabeled

if sh_logger.info: print(f'Total Number of points: [{X_val.shape[0]}]  Mislab
if sh_logger.info: print(f'Percent Mislabeled: [{((y_val != kNN_base_pred).su
if sh_logger.info: print(f'Percent Accurately Labeled: [{kNNAccuractelyLabele

modelsPerformance['PredictAccuracyScore'].append(kNNAccuractelyLabeled)

#print classification report table
targetNames = ["Class{}".format(i) for i in range(n_classes)]
if sh_logger.info: print(f'\n{classification_report(y_val, kNN_base_pred, tar

#print confusion matrix report
cm = confusion_matrix(y_val,kNN_base_pred, labels=[0,1,2,3,4,5,6,7,8,9])
if sh_logger.info: print(f'\nK-Nearest Neighbors Base Classification:\n{cm}')

# plot confusion matrix evaluation
if sh_logger.info: rt.plot_confusion_matrix(cm,classes=[0,1,2,3,4,5,6,7,8,9])
```

4.2.4 Tune Models

5. Algorithm Performance Comparison

Compare the results from the two algorithms. Which one reached higher accuracy? Which one runs faster? Can you explain why?

In [ ]:
```
# hw6 model performance results
# dtc, cnb, gnb - with cross fold validation
modelsPerformance['Name'].append('dtc_cv'); modelsPerformance['TestAccuracySc
modelsPerformance['Name'].append('cnb_cv'); modelsPerformance['TestAccuracySc
modelsPerformance['Name'].append('gnb_cv'); modelsPerformance['TestAccuracySc
```

In [ ]:
```
# model accuracy of predicting labels, and compute time for mode build, score
modelsPerf = pd.DataFrame(modelsPerformance)
modelsPerf['TotalTime'] = modelsPerf['FitTime']+modelsPerf['ScoreTime']+model
modelsPerf.to_csv(f'{dataDir}model_performence_matrix.csv', index=False)
modelsPerf.sort_values(by='PredictAccuracyScore', ascending=False)
```

In [ ]:
```
modelsPerf.sort_values(by=['TotalTime'], ascending=True)
```

In [ ]:
```
# Evaluating the classifiers with a VotingClassifier

from itertools import product
from sklearn.ensemble import VotingClassifier
estimators = [('kNN', kNN_fit), ('rf', rf_base_fit),('svc', svc_fit)]
eclf = VotingClassifier(
                        estimators=estimators,
                        voting='hard',
                        weights=None,
                        n_jobs=None,
                        flatten_transform=True
                        )

eclf.fit(X_train, y_train)
```

In [ ]:
```
# Plotting decision regions
x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))
'''
f, axarr = plt.subplots(2, 2, sharex='col', sharey='row', figsize=(10, 8))


for idx, clf, tt in zip(product([0, 1], [0, 1]),
                        [kNN_fit, rf_base_fit, svc_fit, eclf],
                        ['KNN (k=5)', 'Random Forest (forests=100)',
                         'Kernel SVM', 'Soft Voting']):

    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    axarr[idx[0], idx[1]].contourf(xx, yy, Z, alpha=0.4)
    axarr[idx[0], idx[1]].scatter(X_train[:, 0], X_train[:, 1], c=y,
                                  s=20, edgecolor='k')
    axarr[idx[0], idx[1]].set_title(tt)

plt.show()
'''
```

5. Kaggle Test Results

5.1 Submission File Format

The submission file should be in the following format: For each of the 28000 images in the test set, output a single line containing the ImageId and the digit predicted. For example, if predict that the first image is of a 3, the second image is of a 7, and the third image is of a 8, then the submission file would look like:

ImageId,Label
1,3
2,7
3,8
(27997 more lines)

The evaluation metric for this contest is the categorization accuracy, or the proportion of test images that are correctly classified. For example, a categorization accuracy of 0.97 indicates that, all but 3% of the images have been correctly classified.

In [ ]:

```
%%time
# predictions of unseen test set for submission
t = 0.0
with rt.elapsed_timer() as elapsed:
    kNN_base_submit_pred = kNN_base.predict(X_test)
    t = elapsed()
    if sh_logger.debug: print(f'kNN Model Predict Time, Unseen Submission Tes
```

In [ ]:

```
if sh_logger.debug: print(f'kNN Classification Submission Test size: {kNN_bas

sPred = pd.DataFrame(X_test)
#sub_predictions[1:40]
sample_sub['Label'] = kNN_base_submit_pred
sample_sub.to_csv(f'{outputDir}kNN_submission.csv', index=False)
sample_sub.head(10)
```