1.a

```
[4] imp = SimpleImputer(missing_values = np.NaN, strategy='median')
    df1 = imp.fit_transform(df_missing)
    MAPE1 = MAPE(df_original['displacement'], df1[:,2:3])
    MAPE2 = MAPE(df_original['horsepower'], df1[:,3:4])
    print(MAPE1,MAPE2)
```

```
0.0635665025588773 0.05874415132782256
```

```
[5] imp = KNNImputer(missing_values = np.NaN, n_neighbors=2, weights='uniform')
    df2 = imp.fit_transform(df_missing)
    MAPE1 = MAPE(df_original['displacement'], df2[:,2:3])
    MAPE2 = MAPE(df_original['horsepower'], df2[:,3:4])
    print(MAPE1,MAPE2)
```

```
0.026509356368597293 0.02962845166199372
```

```
[6] imp = IterativeImputer(missing_values=np.nan,max_iter=10,random_state = 0)
    df3 = imp.fit_transform(df_missing)
    MAPE1 = MAPE(df_original['displacement'], df3[:,2:3])
    MAPE2 = MAPE(df_original['horsepower'], df3[:,3:4])
    print(MAPE1,MAPE2)
```

```
0.013151120277339733 0.01935303128971337
```

1.b
According to the MAPE between true and filled value across three methods, the iterative imputer performs the best

1.c
Randomly select a subset of the data without missing value, and make a few scatter plots to observe the relationship between variables with and without missing values. With the approximate relationship in advance, we can use different imputers to fill the missing values and make the same scatter plots again. If the results shows a similar pattern with the scatter plot without missing values, that specific imputer might be a good option.

2.a

```python
df_inspections_train_topics = df_inspections_train.merge(right=biz_topics, on='yelp_business_id')
df_inspections_test_topics = df_inspections_test.merge(right=biz_topics, on='yelp_business_id')
df_inspections_train_topics.head()
```

dba	inspectionid	zipcode	cuisine	venue	chain_restaurant	inspdate	score	yelp_business_id	review_count	stars	topic_0	topic_1	topic_2	topic_3	topic_4
EETGREEN	1270182	10012	Salads	Restaurant (no bar)	1	2015-09-10	12	sweetgreen-new-york-4	128	4.0	0.242166	0.381730	0.033855	0.073409	0.268839
EETGREEN	1276837	10012	Salads	Restaurant (no bar)	1	2016-09-06	20	sweetgreen-new-york-4	128	4.0	0.242166	0.381730	0.033855	0.073409	0.268839
PANERA BREAD	1139571	10028	American	Restaurant (no bar)	1	2014-08-13	17	panera-bread-new-york-6	126	3.0	0.136304	0.450557	0.306742	0.020602	0.085795
PANERA BREAD	1144197	10028	American	Restaurant (no bar)	1	2014-09-12	40	panera-bread-new-york-6	126	3.0	0.136304	0.450557	0.306742	0.020602	0.085795
PANERA BREAD	1153664	10028	American	Restaurant (no bar)	1	2014-09-16	2	panera-bread-new-york-6	126	3.0	0.136304	0.450557	0.306742	0.020602	0.085795

```python
print(lda.perplexity(tf))
```

```
543.054353726795
```

2.b&c

```python
pip = make_pipeline(PolynomialFeatures(degree = 2, include_bias= False),StandardScaler(with_mean=False),LinearRegression())
X_train = df_inspections_train_topics[['chain_restaurant','stars','topic_0','topic_1','topic_2','topic_3','topic_4']]
y_train = df_inspections_train_topics['score']
pip.fit(X_train,y_train)
preds = pip.predict(X_train)
Mean_Squared_Error = MSE(y_train,preds)
print(Mean_Squared_Error)
#scores = cross_val_score(pip, X_train, y_train, cv=10,scoring = 'neg_mean_squared_error')
#print(scores)
```

```
108.88896599583623
```

2.d

```python
[38] print(lda.perplexity(tf))
```

```
495.42668497924507
```

```python
df_inspections_train_topics = df_inspections_train.merge(right=biz_topics, on='yelp_business_id')
df_inspections_test_topics = df_inspections_test.merge(right=biz_topics, on='yelp_business_id')
df_inspections_train_topics.head()
```

amis	dba	inspectionid	zipcode	cuisine	venue	chain_restaurant	inspdate	score	yelp_business_id	review_count	stars	topic_0	topic_1	topic_2	topic_3
17275	SWEETGREEN	1270182	10012	Salads	Restaurant (no bar)	1	2015-09-10	12	sweetgreen-new-york-4	128	4.0	0.344045	0.102942	0.215969	0.337044
17275	SWEETGREEN	1276837	10012	Salads	Restaurant (no bar)	1	2016-09-06	20	sweetgreen-new-york-4	128	4.0	0.344045	0.102942	0.215969	0.337044
71229	PANERA BREAD	1139571	10028	American	Restaurant (no bar)	1	2014-08-13	17	panera-bread-new-york-6	126	3.0	0.322586	0.386594	0.103658	0.187162
71229	PANERA BREAD	1144197	10028	American	Restaurant (no bar)	1	2014-09-12	40	panera-bread-new-york-6	126	3.0	0.322586	0.386594	0.103658	0.187162
71229	PANERA BREAD	1153664	10028	American	Restaurant (no bar)	1	2014-09-16	2	panera-bread-new-york-6	126	3.0	0.322586	0.386594	0.103658	0.187162

```python
pip = make_pipeline(PolynomialFeatures(degree = 2, include_bias= False),StandardScaler(with_mean=False),LinearRegression())
X_train = df_inspections_train_topics[['chain_restaurant','stars','topic_0','topic_1','topic_2','topic_3']]
y_train = df_inspections_train_topics['score']
pip.fit(X_train,y_train)
preds = pip.predict(X_train)
Mean_Squared_Error = MSE(y_train,preds)
print(Mean_Squared_Error)
#scores = cross_val_score(pip, X_train, y_train, cv=10,scoring = 'neg_mean_squared_error')
#print(scores)
```

```
109.55490978487161
```

(0.01,0.01,5) has lower MSE on the training data and more accurate prediction, but it has a higher perplexity.

2.e

(0.01,0.01,5)

```
pip = make_pipeline(PolynomialFeatures(degree = 2, include_bias= False),StandardScaler(with_mean=False),LinearRegression())
X_train = df_inspections_train_topics[['chain_restaurant','stars','topic_0','topic_1','topic_2','topic_3','topic_4']]
y_train = df_inspections_train_topics['score']
#pip.fit(X_train,y_train)
#preds = pip.predict(X_train)
#Mean_Squared_Error = MSE(y_train,preds)
#print(Mean_Squared_Error)
scores = cross_val_score(pip, X_train, y_train, cv=10,scoring = 'neg_mean_squared_error')
print(scores)
print(np.array(scores).mean())
```

```
[-116.80044746 -117.66867242 -110.75689574 -103.47586588 -116.10734883
 -112.72386473 -105.30945885 -126.84450277 -109.79577721 -118.00694444]
-113.74897783272056
```

(1,1,4)

```
pip = make_pipeline(PolynomialFeatures(degree = 2, include_bias= False),StandardScaler(with_mean=False),LinearRegression())
X_train = df_inspections_train_topics[['chain_restaurant','stars','topic_0','topic_1','topic_2','topic_3']]
y_train = df_inspections_train_topics['score']
#pip.fit(X_train,y_train)
#preds = pip.predict(X_train)
#Mean_Squared_Error = MSE(y_train,preds)
#print(Mean_Squared_Error)
scores = cross_val_score(pip, X_train, y_train, cv=10,scoring = 'neg_mean_squared_error')
print(scores)
print(np.array(scores).mean())
```

```
[-114.52821882 -123.94504186 -114.26241339 -101.19285301 -116.10685673
 -101.00694444 -104.69603588 -121.52618634 -109.66536458 -115.49884259]
-112.24287576627287
```

My conclusion changed that parameters (1,1,4) has a lower MSE

3.a

Soved for:

3b1 + 3b2 = 12

b1 + b2 = 4

5b1 + 5b2 = 20

The functions holds if b1 + b2 = 4 and b0 = 0:

A possible sets of parameters can be (b1 = 2, b2 =2, b0 = 0)

3.b

MSE with Lasso = (12 - (3*a + 3*b))**2 + abs(a) + abs(b) + (4- (a + b))**2 + abs(a) + abs(b) + (20 - (5a + 5b))**2 + abs(a) + abs(b)

Let MSE = 0, our objective is to minimize abs(a) + abs(b) while a + b equals to 4.

B1 = a can be any number from 0 to 4, b2 = 4 - a, b0 = 0

(B1 = a, B2 = 4 - a, B0 = 0) where a is in [0,4]

3.c
MSE with Ridge = (12 - (3*a + 3*b))**2 + a**2 + b**2 + (4- (a + b))**2 + a**2 + b**2 + (20 - (5a + 5b))**2 + a**2 + b**2
Let MSE = 0, our objective is to let minimize a**2 + b**2 while a + b equals to 4, which is to minimize a**2 + (4-a)**2. Take the derivative of a and get when 4a + 8 =0, the equation is minimized and get a = 2 and b = 2.
(b1 = 2, b2 =2, b0 = 0)

3.d
Ridge regression yields more stable results compared to Lasso. As the loss function in Ridge usually has a global minimum, while Lasso loss function can be minimized with multiple sets of parameters.

4.a
60%

```
[42] prediction = []
     for i,x in data.iterrows():
       x1,x2,x3,target = x.tolist()
       predict = b0 + b1*x1 + b2*x2 + b3*x3
       proba = math.exp(predict)/(1+math.exp(predict))
       prediction.append(proba)
     data['prediction'] = prediction
```

```
correct_count = 0
thres = 0.5
for i,x in zip(data.target,data.prediction):
  if x >= thres:
    val = 1
  else:
    val = 0
  if val == i:
    correct_count += 1
correct_count/len(data.target)

0.6
```

4.b
When threshold equal 0.52, accuracy is maximized at 80%

```
[42] prediction = []
     for i,x in data.iterrows():
       x1,x2,x3,target = x.tolist()
       predict = b0 + b1*x1 + b2*x2 + b3*x3
       proba = math.exp(predict)/(1+math.exp(predict))
       prediction.append(proba)
     data['prediction'] = prediction
```

```
correct_count = 0
thres = 0.52
for i,x in zip(data.target,data.prediction):
  if x >= thres:
    val = 1
  else:
    val = 0
  if val == i:
    correct_count += 1
correct_count/len(data.target)
```

```
0.8
```

5.a

Instance 1 closest neighbor: 2,5 predicted value = (15+20)/2 = 17.5

Instance 5 closest neighbor: 2,3 predicted value = (15+18)/2 = 16.5

5.b

Instance 1 closest neighbor: 2,5,4, predicted value = (23+15+20)/3 = 19.333

Instance 5 closest neighbor: 2,3,1, predicted value = (18+15+7)/3 = 13.333