Finmath 36702, Portfolio Credit Loss, Assignment 1

Joshua Weekes

Package Imports

In [ ]:
```
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm

# restrict floats to be printed with 4 decimal places
pd.options.display.float_format = '{:.4f}'.format

# restrict floats to be 4 significant digits
np.set_printoptions(precision=4)

# how to toggle between these two options
#pd.set_option('display.float_format', '{:.4f}'.format)
#pd.set_option('display.float_format', '{:.2e}'.format)
```

Problem 1

In [ ]:
```
np.random.seed(99)

samples = 10000
record_size = 20
variables = 10

rho = 0.3

cov_matrix = np.eye(variables)
cov_matrix[:variables-1, variables-1] = rho
cov_matrix[variables-1, :variables-1] = rho

decomposed = np.linalg.cholesky(cov_matrix)

ind_normals = np.random.randn(samples, record_size, variables)

correlated_normals = ind_normals @ decomposed.T

corr_normals_swapped = np.swapaxes(correlated_normals, 1, 2)
```

In [ ]:
```
class RunSimulation:
    def __init__(self, samples, record_size, variables, rho_list):
        self.samples = samples
```

```
        self.record_size = record_size
        self.variables = variables
        self.rho_list = rho_list
        self.betas = np.zeros((variables-1, samples))
        self.r_squared = np.zeros((variables-1, samples))
        self.mse = np.zeros((variables-1, samples))
        self.null_model_mse = np.zeros((samples))
        self.stat_sig = np.zeros((variables-1, samples))
        self.labels = [f'X_{i}' for i in range(1, variables)]
        self.labels_betas = [f'Beta_{i}' for i in range(1, variables)]
        self.labels_r_squared = [f'R^2_{i}' for i in range(1, variables)]
        self.labels_mse = [f'MSE_{i}' for i in range(1, variables)]
        self.labels_stat_sig = [f'Stat_Sig_{i/10}' for i in range(1, variables)]

    def gen_normals(self):
        cov_matrix = np.eye(self.variables)
        for i in range(len(self.rho_list)):
            cov_matrix[i, self.variables-1] = self.rho_list[i]
            cov_matrix[self.variables-1, i] = self.rho_list[i]
        decomposed = np.linalg.cholesky(cov_matrix)
        ind_normals = np.random.randn(self.samples, self.record_size, self.variable
        correlated_normals = ind_normals @ decomposed.T
        corr_normals_swapped = np.swapaxes(correlated_normals, 1, 2)
        self.corr_normals_swapped = corr_normals_swapped

    def run_regression(self):
        for i in range(samples):
            y = corr_normals_swapped[i, -1, :]
            self.null_model_mse[i] = ((y - y.mean()) ** 2).mean()

            for j in range(variables-1):
                x = corr_normals_swapped[i, j, :]
                x = sm.add_constant(x)
                model = sm.OLS(y, x)
                self.betas[j, i] = model.fit().params[0]
                self.r_squared[j, i] = model.fit().rsquared
                predictions = model.fit().predict(x)
                self.mse[j, i] = ((predictions - y) ** 2).mean()
                self.stat_sig[j, i] = 1 if model.fit().pvalues[0] < 0.05 else 0

    def create_dataframes(self):
        self.betas_df = pd.DataFrame(self.betas.T, columns=self.labels_betas)
        self.r_squared_df = pd.DataFrame(self.r_squared.T, columns=self.labels_r_sq
        self.mse_df = pd.DataFrame(self.mse.T, columns=self.labels_mse)
        self.stat_sig_df = pd.DataFrame(self.stat_sig.T, columns=self.labels_stat_s

    def combine_dataframes(self, df1, df2):
        self.combined_df = pd.concat([df1, df2], axis=1)

    @staticmethod
    def get_max_r2_beta(row, max_index=10):
        r2_values = row[[f'R^2_{i}' for i in range(1, max_index)]].values
        max_r2_index = np.argmax(r2_values)
        beta_value = row[f'Beta_{max_r2_index + 1}']

        return beta_value
```

```
    @staticmethod
    def get_max_r2_mse(row, max_index=10):
        r2_values = row[[f'R^2_{i}' for i in range(1, max_index)]].values
        max_r2_index = np.argmax(r2_values)
        mse = row[f'MSE_{max_r2_index + 1}']

        return mse

    def problem_1(self):
        self.gen_normals()
        self.run_regression()
        self.create_dataframes()
        self.combine_dataframes(self.betas_df, self.r_squared_df)

        self.combined_df.loc[:, 'Proc_1'] = self.betas_df.loc[:, 'Beta_1']
        self.combined_df.loc[:, 'Proc_2'] = self.combined_df.apply(self.get_max_r2_
        self.combined_df.loc[:, 'Proc_3'] = self.combined_df.apply(self.get_max_r2_

        procedure_labels = [f'Procedure {i}' for i in range(1, 4)]
        procedure_labels

        avg_slope = []
        est_bias = []
        for i in range(3):
            avg_slope.append(self.combined_df.loc[:, f'Proc_{i + 1}'].mean())
            est_bias.append(avg_slope[i] - 0.3)

        self.question_1_results = pd.DataFrame({'Average Slope': avg_slope, 'Estima
                            index=procedure_labels)

        return self.question_1_results


    def problem_2(self):
        mse_combined_df = pd.concat([self.mse_df, self.r_squared_df], axis=1)

        mse_combined_df.loc[:, 'Proc_0'] = self.null_model_mse
        mse_combined_df.loc[:, 'Proc_1'] = self.mse_df.loc[:, 'MSE_1']
        mse_combined_df.loc[:, 'Proc_2'] = mse_combined_df.apply(self.get_max_r2_ms
        mse_combined_df.loc[:, 'Proc_3'] = mse_combined_df.apply(self.get_max_r2_ms

        procedure_labels_q2 = [f'Procedure {i}' for i in range(4)]
        procedure_labels_q2

        avg_mse = []
        for i in range(4):
            avg_mse.append(mse_combined_df.loc[:, f'Proc_{i}'].mean())

        question_2_results = pd.DataFrame({'Avereage ESE': avg_mse},
                            index=procedure_labels_q2)

        return question_2_results

    def problem_3(self):
        self.gen_normals()
```

```
        self.run_regression()
        self.create_dataframes()
        sig_stats = self.stat_sig_df == 1
        sig_mse = self.mse_df[sig_stats]
        avg_bias = []
        frac_of_sig_reg = []
        for i in range(10):
            avg_bias.append(sig_mse.loc[:, f'Beta_{i+1}'].mean() - self.rho_list[i]
            frac_of_sig_reg.append(self.stat_sig_df.loc[:, i].count() / samples)
        question_3_results = pd.DataFrame({'Average Bias': avg_bias, 'Fraction of r
                            index=self.labels_stat_sig)
        return question_3_results

    def problem_4(self):
        # Need to run the generate normals, run regression, and create dataframes f
        # for different sample sizes and then combine the results
        self.gen_normals()
        self.run_regression()
        self.create_dataframes()
        sig_stats = self.stat_sig_df == 1
        sig_mse = self.mse_df[sig_stats]
        avg_bias = []
        frac_of_sig_reg = []
        for i in range(10):
            # Change from avg bias to ESE
            avg_bias.append(sig_mse.loc[:, f'Beta_{i+1}'].mean() - self.rho_list[i]
            null_model_mse = self.null_model_mse[sig_stats.loc[:, i]]
        question_4_results = pd.DataFrame({'Avg ESE Null': null_model_mse, 'Avg ESE
                            index=self.labels_stat_sig)
        return question_4_results
```

first iteration

Solution is given 2 ways both with and without using RunSimulation Class

In [ ]:

```
betas = np.zeros((variables-1, samples))
r_squared = np.zeros((variables-1, samples))
mse = np.zeros((variables-1, samples))
null_model_mse = np.zeros((samples))

for i in range(samples):
    y = corr_normals_swapped[i, -1, :]
    null_model_mse[i] = ((y - y.mean()) ** 2).mean()

    for j in range(variables-1):
        x = corr_normals_swapped[i, j, :]
        # x = sm.add_constant(x)
        model = sm.OLS(y, x)
        betas[j, i] = model.fit().params[0]
        r_squared[j, i] = model.fit().rsquared
        predictions = model.fit().predict(x)
        mse[j, i] = ((predictions - y) ** 2).mean()
```

In [ ]:
```
labels = [f'X_{i}' for i in range(1, variables)]
labels_betas = [f'Beta_{i}' for i in range(1, variables)]
labels_r_squared = [f'R^2_{i}' for i in range(1, variables)]
labels_mse = [f'MSE_{i}' for i in range(1, variables)]
```

In [ ]:
```
betas_df = pd.DataFrame(betas.T, columns=labels_betas)
r_squared_df = pd.DataFrame(r_squared.T, columns=labels_r_squared)
mse_df = pd.DataFrame(mse.T, columns=labels_mse)
```

In [ ]:
```
def get_max_r2_beta(row, max_index=10):
    r2_values = row[[f'R^2_{i}' for i in range(1, max_index)]].values
    max_r2_index = np.argmax(r2_values)
    beta_value = row[f'Beta_{max_r2_index + 1}']

    return beta_value
```

In [ ]:
```
combined_df = pd.concat([betas_df, r_squared_df], axis=1)

combined_df.loc[:, 'Proc_1'] = betas_df.loc[:, 'Beta_1']
combined_df.loc[:, 'Proc_2'] = combined_df.apply(get_max_r2_beta, args=[3], axis=1)
combined_df.loc[:, 'Proc_3'] = combined_df.apply(get_max_r2_beta, axis=1)
```

In [ ]:
```
procedure_labels = [f'Procedure {i}' for i in range(1, 4)]
procedure_labels

avg_slope = []
est_bias = []
for i in range(3):
    avg_slope.append(combined_df.loc[:, f'Proc_{i + 1}'].mean())
    est_bias.append(avg_slope[i] - 0.3)

question_1_results = pd.DataFrame({'Average Slope': avg_slope, 'Estimated Bias': es
                       index=procedure_labels)

pd.set_option('display.float_format', '{:.4f}'.format)
question_1_results
```

Out[ ]:

	Average Slope	Estimated Bias
Procedure 1	0.3021	0.0021
Procedure 2	0.4094	0.1094
Procedure 3	0.5678	0.2678

In [ ]:
```
simulations = 10000
record_size = 20
variables = 10
rho_list = [0.3] * (variables-1)

first_problem = RunSimulation(samples, record_size, variables, rho_list)
first_problem.problem_1()
```

Out[ ]:

	Average Slope	Estimated Bias
Procedure 1	-0.0017	-0.3017
Procedure 2	-0.0013	-0.3013
Procedure 3	0.0001	-0.2999

Problem 2

In [ ]:
```
def get_max_r2_mse(row, max_index=10):
    r2_values = row[[f'R^2_{i}' for i in range(1, max_index)]].values
    max_r2_index = np.argmax(r2_values)
    mse = row[f'MSE_{max_r2_index + 1}']

    return mse
```

In [ ]:
```
mse_combined_df = pd.concat([mse_df, r_squared_df], axis=1)

mse_combined_df.loc[:, 'Proc_0'] = null_model_mse
mse_combined_df.loc[:, 'Proc_1'] = mse_df.loc[:, 'MSE_1']
mse_combined_df.loc[:, 'Proc_2'] = mse_combined_df.apply(get_max_r2_mse, args=[3],
mse_combined_df.loc[:, 'Proc_3'] = mse_combined_df.apply(get_max_r2_mse, axis=1)
```

In [ ]:
```
procedure_labels_q2 = [f'Procedure {i}' for i in range(4)]
procedure_labels_q2

avg_mse = []
for i in range(4):
    avg_mse.append(mse_combined_df.loc[:, f'Proc_{i}'].mean())

question_2_results = pd.DataFrame({'Avereage ESE': avg_mse},
                        index=procedure_labels_q2)

pd.set_option('display.float_format', '{:.3e}'.format)
question_2_results
```

Out[ ]:

	Avereage ESE
Procedure 0	9.556e-01
Procedure 1	8.681e-01
Procedure 2	7.992e-01
Procedure 3	6.473e-01

In [ ]:
```
first_problem.problem_2()
```

Out[ ]:

	Avereage ESE
Procedure 0	9.556e-01
Procedure 1	8.231e-01
Procedure 2	7.558e-01
Procedure 3	6.070e-01

not sure why im getting that procedure 3 is the best. Check get r_2 function

Problem 3

Code is currently not working because a problem generating the matrix I think I need to use a 2x2 and create all of the data series separately instead of having the same y for all x's.

In [ ]:
```
rho_list = [i/10 for i in range(10)]

simulations = 10000
record_size = 25
variables = 11

pd.set_option('display.float_format', '{:.4f}'.format)
rho_list
third_problem = RunSimulation(samples, record_size, variables, rho_list)
third_problem.problem_3()
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Cell In[174], line 10
      8 rho_list
      9 third_problem = RunSimulation(samples, record_size, variables, rho_list)
---> 10 third_problem.problem_3()

Cell In[159], line 115, in RunSimulation.problem_3(self)
    114 def problem_3(self):
--> 115     self.gen_normals()
    116     self.run_regression()
    117     self.create_dataframes()

Cell In[159], line 23, in RunSimulation.gen_normals(self)
     21     cov_matrix[i, self.variables-1] = self.rho_list[i]
     22     cov_matrix[self.variables-1, i] = self.rho_list[i]
---> 23 decomposed = np.linalg.cholesky(cov_matrix)
     24 ind_normals = np.random.randn(self.samples, self.record_size, self.variable
s)
     25 correlated_normals = ind_normals @ decomposed.T

File c:\Users\joshw\anaconda3\Lib\site-packages\numpy\linalg\linalg.py:779, in chole
sky(a)
    777 t, result_t = _commonType(a)
    778 signature = 'D->D' if isComplexType(t) else 'd->d'
--> 779 r = gufunc(a, signature=signature, extobj=extobj)
    780 return wrap(r.astype(result_t, copy=False))

File c:\Users\joshw\anaconda3\Lib\site-packages\numpy\linalg\linalg.py:115, in _rais
e_linalgerror_nonposdef(err, flag)
    114 def _raise_linalgerror_nonposdef(err, flag):
--> 115     raise LinAlgError("Matrix is not positive definite")

LinAlgError: Matrix is not positive definite
```

Problem 4

In [ ]:
```
rho_list = [0.2]

simulations = 10000
record_size_list = [10, 20, 30, 40, 50]
variables = 2


rho_list
third_problem = RunSimulation(samples, record_size, variables, rho_list)
third_problem.problem_4()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File c:\Users\joshw\anaconda3\Lib\site-packages\pandas\core\indexes\base.py:3791, in
Index.get_loc(self, key)
   3790 try:
-> 3791     return self._engine.get_loc(casted_key)
   3792 except KeyError as err:

File index.pyx:152, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:181, in pandas._libs.index.IndexEngine.get_loc()

File pandas\_libs\hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.PyObjec
tHashTable.get_item()

File pandas\_libs\hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjec
tHashTable.get_item()

KeyError: 'Beta_1'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[175], line 10
      8 rho_list
      9 third_problem = RunSimulation(samples, record_size, variables, rho_list)
---> 10 third_problem.problem_4()

Cell In[159], line 141, in RunSimulation.problem_4(self)
    138 frac_of_sig_reg = []
    139 for i in range(10):
    140     # Change from avg bias to ESE
--> 141     avg_bias.append(sig_mse.loc[:, f'Beta_{i+1}'].mean() - self.rho_list[i])
    142     null_model_mse = self.null_model_mse[sig_stats.loc[:, i]]
    143 question_4_results = pd.DataFrame({'Avg ESE Null': null_model_mse, 'Avg ESE
Reg': avg_bias},
    144                     index=self.labels_stat_sig)

File c:\Users\joshw\anaconda3\Lib\site-packages\pandas\core\indexing.py:1147, in _Lo
cationIndexer.__getitem__(self, key)
   1145     if self._is_scalar_access(key):
   1146         return self.obj._get_value(*key, takeable=self._takeable)
-> 1147     return self._getitem_tuple(key)
   1148 else:
   1149     # we by definition only have the 0th axis
   1150     axis = self.axis or 0

File c:\Users\joshw\anaconda3\Lib\site-packages\pandas\core\indexing.py:1330, in _Lo
cIndexer._getitem_tuple(self, tup)
   1328 with suppress(IndexingError):
   1329     tup = self._expand_ellipsis(tup)
-> 1330     return self._getitem_lowerdim(tup)
   1332 # no multi-index, so validate all of the indexers
   1333 tup = self._validate_tuple_indexer(tup)

File c:\Users\joshw\anaconda3\Lib\site-packages\pandas\core\indexing.py:1039, in _Lo
```

```
cationIndexer._getitem_lowerdim(self, tup)
   1035 for i, key in enumerate(tup):
   1036     if is_label_like(key):
   1037         # We don't need to check for tuples here because those are
   1038         #  caught by the _is_nested_tuple_indexer check above.
-> 1039         section = self._getitem_axis(key, axis=i)
   1041         # We should never have a scalar section here, because
   1042         #  _getitem_lowerdim is only called after a check for
   1043         #  is_scalar_access, which that would be.
   1044         if section.ndim == self.ndim:
   1045             # we're in the middle of slicing through a MultiIndex
   1046             # revise the key wrt to `section` by inserting an _NS

File c:\Users\joshw\anaconda3\Lib\site-packages\pandas\core\indexing.py:1393, in _Lo
cIndexer._getitem_axis(self, key, axis)
   1391 # fall thru to straight lookup
   1392 self._validate_key(key, axis)
-> 1393 return self._get_label(key, axis=axis)

File c:\Users\joshw\anaconda3\Lib\site-packages\pandas\core\indexing.py:1343, in _Lo
cIndexer._get_label(self, label, axis)
   1341 def _get_label(self, label, axis: AxisInt):
   1342     # GH#5567 this will fail if the label is not present in the axis.
-> 1343     return self.obj.xs(label, axis=axis)

File c:\Users\joshw\anaconda3\Lib\site-packages\pandas\core\generic.py:4222, in NDFr
ame.xs(self, key, axis, level, drop_level)
   4220 if axis == 1:
   4221     if drop_level:
-> 4222         return self[key]
   4223     index = self.columns
   4224 else:

File c:\Users\joshw\anaconda3\Lib\site-packages\pandas\core\frame.py:3893, in DataFr
ame.__getitem__(self, key)
   3891 if self.columns.nlevels > 1:
   3892     return self._getitem_multilevel(key)
-> 3893 indexer = self.columns.get_loc(key)
   3894 if is_integer(indexer):
   3895     indexer = [indexer]

File c:\Users\joshw\anaconda3\Lib\site-packages\pandas\core\indexes\base.py:3798, in
Index.get_loc(self, key)
   3793     if isinstance(casted_key, slice) or (
   3794         isinstance(casted_key, abc.Iterable)
   3795         and any(isinstance(x, slice) for x in casted_key)
   3796     ):
   3797         raise InvalidIndexError(key)
-> 3798     raise KeyError(key) from err
   3799 except TypeError:
   3800     # If we have a listlike key, _check_indexing_error will raise
   3801     #  InvalidIndexError. Otherwise we fall through and re-raise
   3802     #  the TypeError.
   3803     self._check_indexing_error(key)

KeyError: 'Beta_1'
```

Extra methods I was testing out

In [ ]:
```
class RandomDraws():
    def __init__(self, rho, n=25, p=2):
        self.n = n
        self.p = p
        self.rho = rho
        self.cov_matrix = np.eye(p)
        self.cov_matrix[:p-1, p-1] = rho
        self.cov_matrix[p-1, :p-1] = rho
        self.decomposed = np.linalg.cholesky(self.cov_matrix)
        self.ind_normals = np.random.randn(n, p)
        self.correlated_normals = self.ind_normals @ self.decomposed.T
        self.correlated_normals_swapped = np.swapaxes(self.correlated_normals, 0, 1

    def get_draws(self):
        return self.correlated_normals_swapped
```

In [ ]:
```
class RandomDrawFactory():
    def __init__(self, samples, rho, n=25, p=2):
        self.rho = rho
        self.n = n
        self.p = p
        self.samples = samples

    def get_draws(self):
        result = []
        for i in range(self.samples):
            result.append(RandomDraws(self.rho, self.n, self.p).get_draws())
        return result
```

In [ ]:
```
class RandomDrawProcessor():
    def __init__(self, samples, rho_list=[0], n=25, p=2):
        self.samples = samples
        self.rho_list = rho_list
        self.n = n
        self.p = p
        self.factory = None
        self.draws = None

    def generate_draws(self):
        for rho in self.rho_list:
            self.factory = RandomDrawFactory(self.samples, rho, self.n, self.p)
            self.draws = self.factory.get_draws()

    def get_results(self):
        betas = np.zeros((self.p-1, self.samples))
        r_squared = np.zeros((self.p-1, self.samples))
        mse = np.zeros((self.p-1, self.samples))
        stat_sig = np.zeros((self.p-1, self.samples))

        for i in range(self.samples):
            y = self.draws[i][-1, :]
```

```
            null_model_mse[i] = ((y - y.mean()) ** 2).mean()

            for j in range(self.p-1):
                x = self.draws[i][j, :]
                model = sm.OLS(y, x)
                betas[j, i] = model.fit().params[0]
                r_squared[j, i] = model.fit().rsquared
                predictions = model.fit().predict(x)
                stat_sig[j, i] = model.fit().pvalues[0]
                mse[j, i] = ((predictions - y) ** 2).mean()

        return betas, r_squared, mse, stat_sig
```

In [ ]:
```
class RegressionSimulator():
    def __init__(self, sample_size, num_simulations):
        self.sample_size = sample_size
        self.num_simulations = num_simulations
        self.results = None

    def simulate_regressions(self, rho):
        betas = []
        sig_count = 0

        for _ in range(self.num_simulations):
            X, Y = self.generate_samples(rho, self.sample_size)
            slope, p_value = self.run_regression(X, Y)
            betas.append(slope)
            if p_value < 0.05:
                sig_count += 1

        mean_beta = np.mean(betas)
        return mean_beta, sig_count / self.num_simulations

    @staticmethod
    def generate_samples(rho, sample_size):
        X = np.random.randn(sample_size, 2)
        Y = X @ np.array([1, rho]) + np.random.randn(sample_size)
        return X, Y
```

In [ ]:
```
rho_list = [i/10 for i in range(10)]
rho_list
```

```
[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
```

In [ ]:
```
draw_list = []

for rho in rho_list:
    draws = RandomDrawFactory(10000, rho, 25, 2)
    draw_list.append(draws)
```