Problem Statement

The data given is of the mutual funds in USA. The objective of this problem is to predict the 'basis point spread' over AAA bonds i.e. feature 'bonds_aaa' against each Serial Number.

Basis Point Spread indicates the additional return a mutual fund would give over the AAA rated bonds.

About the Dataset

The data given is of the mutual funds in USA. Following is the brief description of the features in this data

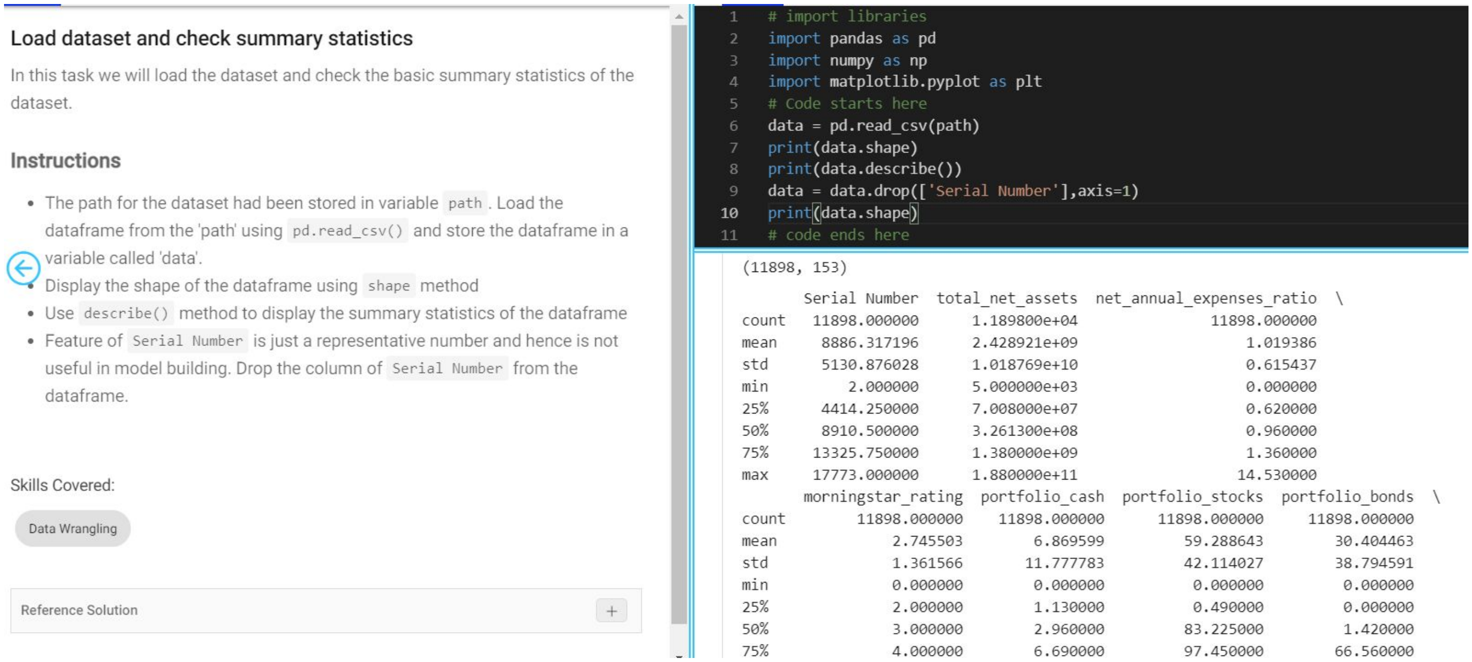
- `Fund Symbol`: Uniques symbol for the mutual fund used for representing it on the bourses
- `Fund Name`: Full name of the mutual fund scheme
- `Category`: Investment category of the mutual fund
- `Fund Family`: Asset management company to which mutual fund belongs to
- `Investment`: Type of investment of the mutual fund scheme
- `Size`: Size of the mutual fund based on the total net assets
- `Total net assets`: Total assets under management for the mutual fund scheme
- `Currency`: Currency in which the investments of the mutual fund are held
- `Net Annual Expense Ratio`: Expense ratio is the fee that the asset management company charges to the clients as a percentage of the total assets.
- `Morningstar Rating`: This is the overall fund rating given by the rating agency Morning Star. The rating is on the scale of 1 to 5 where 5 is the best.
- `Inception Date`: The date on which the mutual fund scheme was started.
- `portfolio`: percentage of total assets invested in the investment instrument.
- `sectors`: percentage of equity assets invested in the sector
- `Morningstar Return Rating`: Fund rating based on returns by the rating agency Morning Star.The rating is on the scale of 1 to 5 where 5 is the best. `Returns_ytd`: Year to date return of the mutual fund.
- `retruns`: Annual return of the mutual fund for the respective year
- `Morningstar Risk Rating`: Fund rating based on risk of the mutual fund by the rating agency Morning Star.The rating is on the scale of 1 to 5 where 5 is the best.
- `Alpha 3y`: 3year average alpha of the mutual fund.
- `Beta 3y`: 3year average beta of the mutual fund.
- `Mean Annual Return 3y`: 3year mean annual return
- `Standard Deviation 3y`: Standard deviation of returns over three years.
- `Sharpe Ratio 3y`: 3year average Sharpe ratio of the mutual fund.
- `bonds_*`: Basis point spread over the bonds for the mutual fund.

The original data contains many non numerical features and missing values. We will be learning how to handle these cases in future concepts, for this project we have processed the data for you so that you can concentrate on building model.

Why solve this project?

After completing this project you will have better understanding of how to apply linear model using `GridsearchCV`.

- Chi square contengency test
- Box plot
- Linear regression
- GridsearchCV
- Ridge and Lasso Regressor



Load dataset and check summary statistics

In this task we will load the dataset and check the basic summary statistics of the dataset.

Instructions

- The path for the dataset had been stored in variable `path`. Load the dataframe from the 'path' using `pd.read_csv()` and store the dataframe in a variable called 'data'.
- Display the shape of the dataframe using `shape` method
- Use `describe()` method to display the summary statistics of the dataframe
- Feature of `Serial Number` is just a representative number and hence is not useful in model building. Drop the column of `Serial Number` from the dataframe.

Skills Covered:

Data Wrangling

Reference Solution

```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# Code starts here
data = pd.read_csv(path)
print(data.shape)
print(data.describe())
data = data.drop(['Serial Number'],axis=1)
print(data.shape)
# code ends here
```

```
(11898, 153)
       Serial Number  total_net_assets  net_annual_expenses_ratio  \
count   11898.000000      1.189800e+04               11898.000000
mean     8886.317196      2.428921e+09                   1.019386
std      5130.876028      1.018769e+10                   0.615437
min         2.000000      5.000000e+03                   0.000000
25%      4414.250000      7.008000e+07                   0.620000
50%      8910.500000      3.261300e+08                   0.960000
75%     13325.750000      1.380000e+09                   1.360000
max     17773.000000      1.880000e+11                  14.530000
       morningstar_rating  portfolio_cash  portfolio_stocks  portfolio_bonds  \
count        11898.000000    11898.000000      11898.000000     11898.000000
mean             2.745503        6.869599         59.288643        30.404463
std              1.361566       11.777783         42.114027        38.794591
min              0.000000        0.000000          0.000000         0.000000
25%              2.000000        1.130000          0.490000         0.000000
50%              3.000000        2.960000         83.225000         1.420000
75%              4.000000        6.690000         97.450000        66.560000
```

Hypothesis Testing

Another thing bank suspects is that there is a strong assosciation between `morningstar return rating` and `morningstar risk rating`. Now let's check it by using null hypothesis. Since both are categorical columns, we will do chi-square test to test the same.

- Null Hypothesis : Both the features are independent from each other.
- Alternative Hypothesis : Both features are dependent on each other.

- Create a variable `'return_rating'` which is the value counts of `morningstar_return_rating`
- Create a variable `'risk_raing'` which is the value counts of `morningstar_risk_rating`
- Concat `'return_rating.transpose()'` and `'risk_rating.transpose()'` along `axis=1` with `keys=['return','risk']` and store it in a variable called `'observed'`
- Apply `"chi2_contingency()"` on `'observed'` and store the result in variables named `chi2, p, dof, ex` respectively.
- Compare `chi2` with `critical_value`(given)
- If chi-squared statistic exceeds the `critical_value`, reject the null hypothesis that the both features are independent from each other, else null hypothesis cannot be rejected.

```python
#Importing header files
from scipy.stats import chi2_contingency
import scipy.stats as stats

#Critical value
critical_value = stats.chi2.ppf(q = 0.95, # Find the critical value for 95% confidence*
                      df = 11)   # Df = number of variable categories(in purpose) - 1

# Code starts here
return_rating = data['morningstar_return_rating'].value_counts()
print(return_rating)
risk_rating = data['morningstar_risk_rating'].value_counts()
print(risk_rating)
observed =
pd.concat([return_rating.transpose(),risk_rating.transpose()],axis=1,keys=['return','ri
sk'])
print(observed)
chi2, p, dof, ex = chi2_contingency(observed)
print("p value")
print(p)
print("Chi Statistic")
print(chi2)
if chi2 > critical_value:
   print("Null Hypothesis is Rejected")
else:
   print("Null Hypothesis is Accepted")
# Code ends here
```

Output:

```
3    3892
4    2628
2    2422
0    1236
5     956
1     764
Name: morningstar_return_rating, dtype: int64
3    3845
4    2614
2    2218
0    1236
5    1080
1     905
Name: morningstar_risk_rating, dtype: int64
  return  risk
3    3892  3845
4    2628  2614
2    2422  2218
0    1236  1236
5     956  1080
```

```
1      764    905
p value
2.5889934498733718e-05
Chi Statistic
28.75585318206671
Null Hypothesis is Rejected
```

Remove correlated features

As we have learned earlier one of the assumptions of Linear Regression model is that the independent features should not be correlated to each other. In this task we will find the features that have a correlation higher that 0.75 and remove the same so that the assumption for linear regression model is satisfied.

Instructions

- Use `corr()` method to calculate the correlation between all the features. Use `abs()` method to get only the absolute values of the correlation. Store the values in dataframe `correlation`
- Print `correlation`.
- As you can observe we get a dataframe with correlation calculated for each pair, this dataframe needs some transformation to extract the features with correlation greater than 0.75
- Use `unstack()` method on `correlation` and store the same in `us_correlation`
- Use `sort_value()` method to sort the correlation with `ascending=False` parameter. Store the sorted series in `us_correlation`
- Apply a filter to `us_correlation` to extract the pairs with correlatio higher that 0.75 but less than 1. Store the filtered values to `max_correlated`
- You can observe that we have 4 pairs of features which have correlation higher than 0.75. Based on this information drop the features of `morningstar_rating`, `portfolio_stocks`, `category_12` and `sharpe_ratio_3y` from the `data`

```
# Code starts here
correlation = abs(data.corr())
print(correlation)
us_correlation = correlation.unstack()
print(us_correlation)
us_correlation = us_correlation.sort_values(ascending = False)
print(us_correlation)
max_correlated = us_correlation[(us_correlation>0.75) & (us_correlation<1)]
print(max_correlated)
data.drop(['morningstar_rating', 'portfolio_stocks', 'category_12'
,'sharpe_ratio_3y'],axis=1,inplace=True)
print(data.shape)
# code ends here
```

Outlier check!

Now let's plot the box plot for `price_earning` and `net_annual_expenses`. So that we can check the outlier.

Instructions:

- Create two subplots with axes as `'ax_1'`, `'ax_2'`.
- Plot a boxplot of `price_earning` column using `ax_1`. Set axis title as `price_earning`.
- Plot a boxplot of `net_annual_expenses_ratio` column using `ax_2`. Set axis title as `net_annual_expenses_ratio`

Skills Covered:

Visualization

Reference Solution +

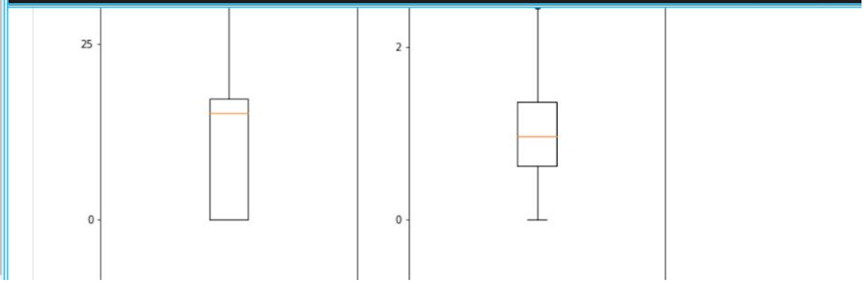
```python
# Code starts here
fig, (ax_1, ax_2) = plt.subplots(1, 2, figsize=(10,25))
ax_1.boxplot(data['price_earning'])
ax_1.set_title("price_earning")
ax_2.boxplot(data['net_annual_expenses_ratio'])
ax_2.set_title("net_annual_expenses_ratio")
# code ends here
```

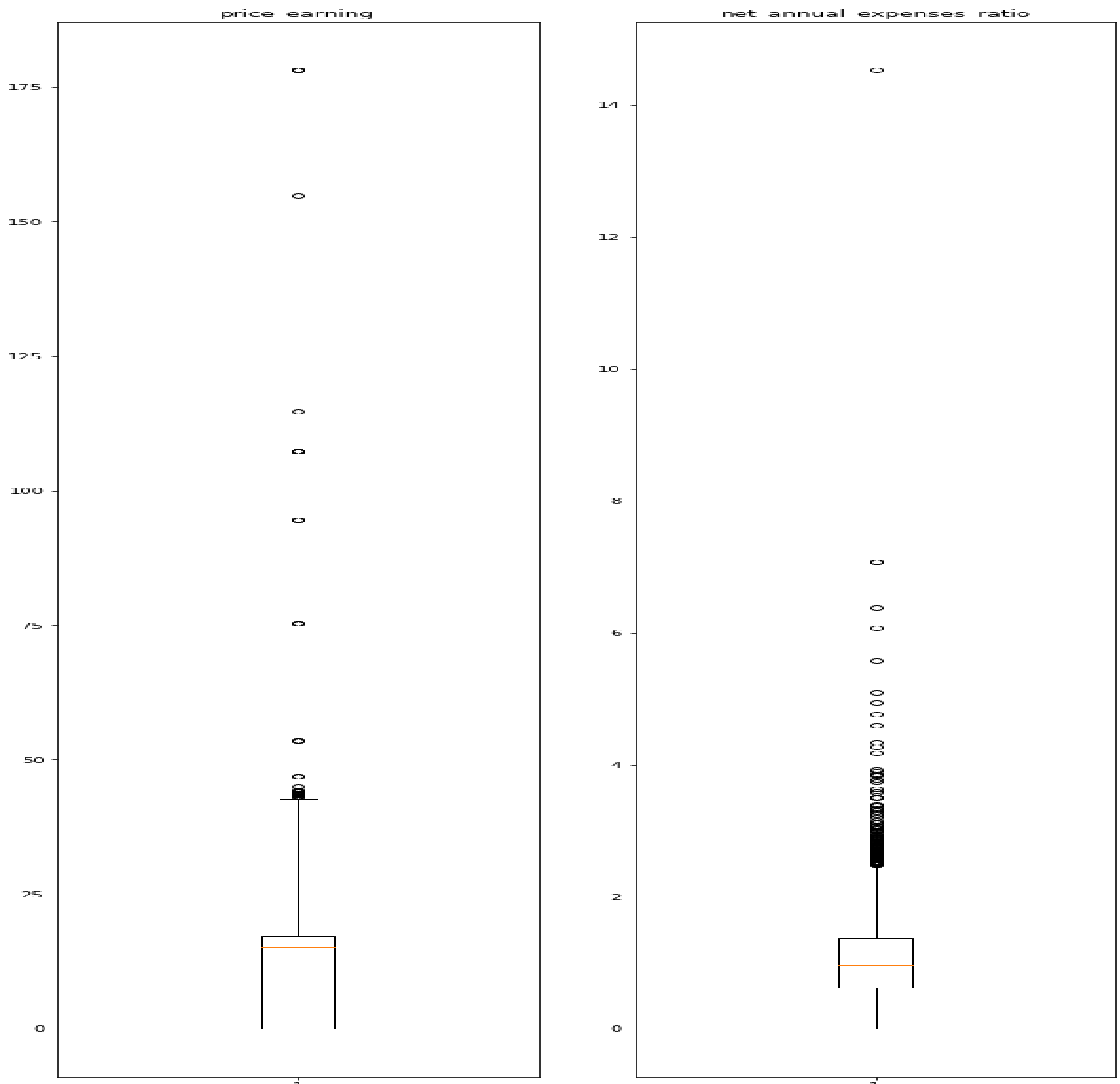
Congrats!

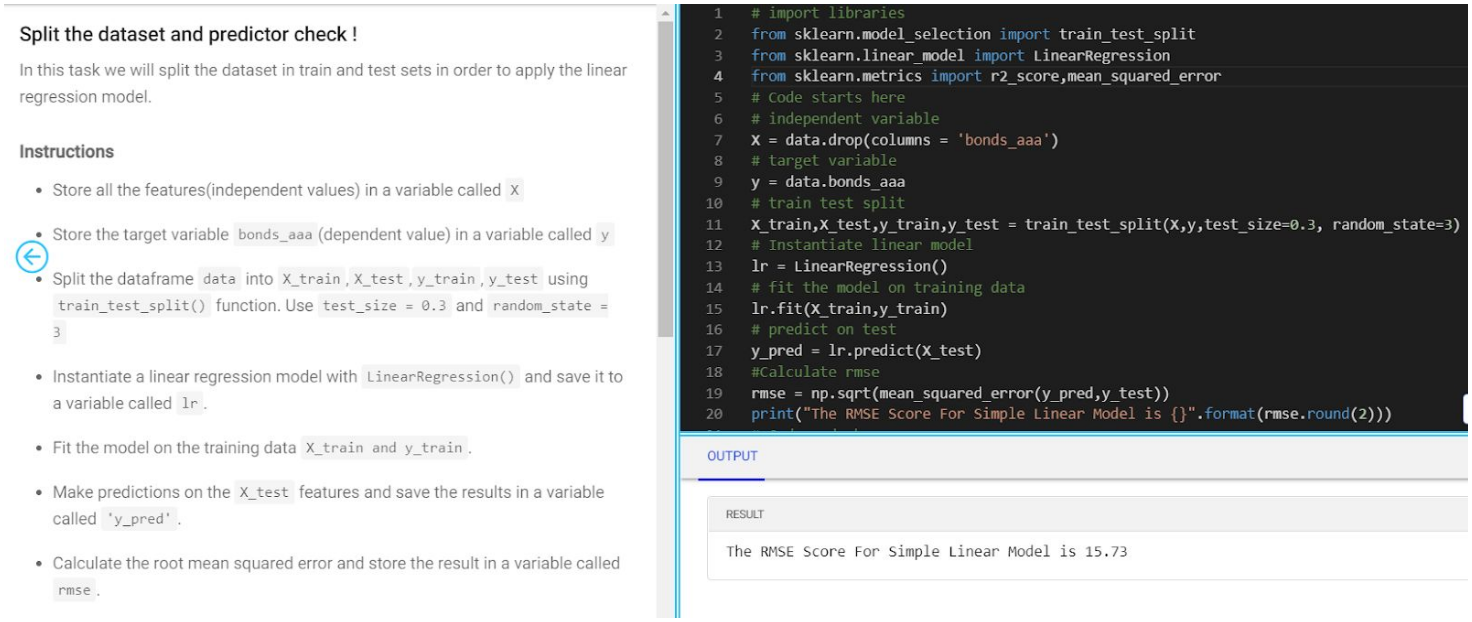
You have successfully plotted the box plot. As you can see that average price earning is `$20K` per month and net anual expense ratio is `1.5`.

CONTINUE

TRY IT







Predictor check using GridsearchCV

Regularization is a technique used to prevent overfitting in datasets. Sometime complex model may not perform well in test data due to over fitting. We need to choose the right model in between simple and complex model. Regularization helps to choose preferred model complexity, so that model is better at predicting.

Now in this task let's predict the bonds_aaa using lasso regressor and ridge regressor with the help of gridsearch cv , check is there any improvement in the prediction.

Instructions:

- Instantiate a model with Ridge() with and save it to a variable called `ridge_model`.
- Apply `GridSearchCV` as `GridSearchCV(estimator=ridge_model, param_grid=dict(alpha=ridge_lambdas))` and store it in variable `ridge_grid`.
- Fit the `ridge_grid` on the training data `X_train` and `y_train`.
- Calculate the `rmse` score for the above model and store the value in `ridge_rmse`
- Instantiate a model with lasso() with and save it to a variable called `lasso_model`.
- Apply `GridSearchCV` as `GridSearchCV(estimator=lasso_model, param_grid=dict(alpha=lasso_lambdas))` and store it in variable `lasso_grid`.
- Fit the `lasso_grid` on the training data `X_train` and `y_train`.
- Calculate the `rmse` score for above model and store the value in `lasso_rmse`

```
# import libraries
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.linear_model import Ridge,Lasso

# regularization parameters for grid search
ridge_lambdas = [0.01, 0.03, 0.06, 0.1, 0.3, 0.6, 1, 3, 6, 10, 30, 60]
```

```python
lasso_lambdas = [0.0001, 0.0003, 0.0006, 0.001, 0.003, 0.006, 0.01, 0.03, 0.06, 0.1,
0.3, 0.6, 1]

# Code starts here

# Instantiate ridge models
ridge_model = Ridge()

# apply ridge model
ridge_grid = GridSearchCV(estimator=ridge_model, param_grid=dict(alpha=ridge_lambdas))
ridge_grid.fit(X_train, y_train)

# make predictions
ridge_pred = ridge_grid.predict(X_test)
ridge_rmse = np.sqrt(mean_squared_error(ridge_pred, y_test))
print(ridge_rmse)

# Instantiate lasso models
lasso_model = Lasso()
# apply lasso model
lasso_grid = GridSearchCV(estimator=lasso_model, param_grid=dict(alpha=lasso_lambdas))
lasso_grid.fit(X_train, y_train)

# make predictions
lasso_pred = lasso_grid.predict(X_test)
lasso_rmse = np.sqrt(mean_squared_error(lasso_pred, y_test))
print(lasso_rmse)

# Code ends here
```

```
RMSE Score For Lasso Model is 15.719153628852963
RMSE Score For Ridge Model is 15.720131026226952
```