CE802 REPORT

Jhansi Rani Choutapalem

Word count: 1495

Introduction

Travel insurance is one business sector where there is an abundance of data, using machine learning techniques companies wants to take effective decisions by utilizing the data in their hands in order to improve their market strategy, attract customers with discounted deals and increase their profits by spending their funds wisely.

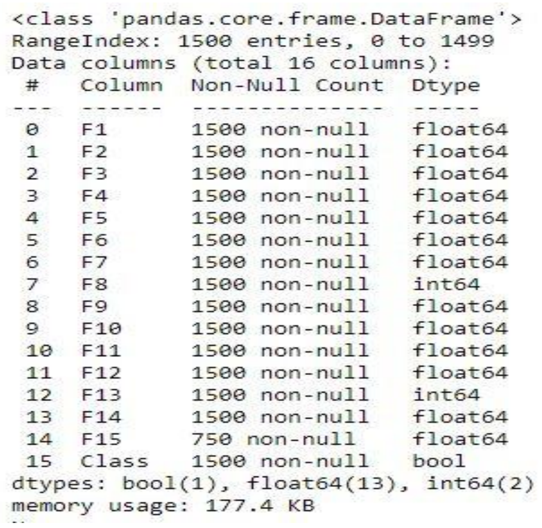
There are lots of machine learning algorithms available now a days. But, effective use of any model built using these algorithms requires appropriate preparation of the input data and tuning hyperparameters of the model. This report focusses on two prediction tasks classification and regression, the details of which are mentioned below.

Part 2 (Classification)

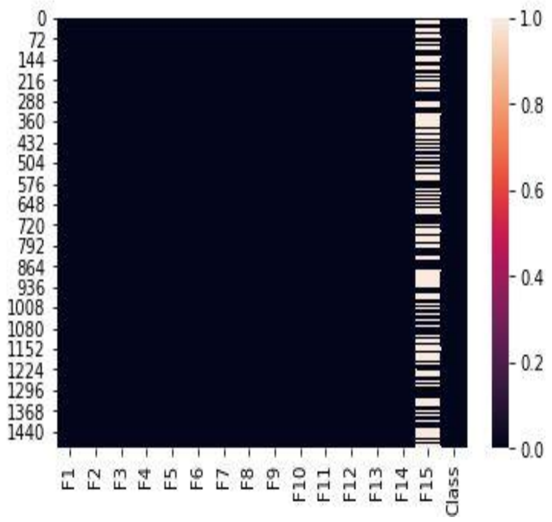
The prediction task in part 2 of assignment involves classifying the customer into two categories; True or False, True if the customer files a claim or False if the customer doesn't file a claim using the data of past travellers given by the travel insurance company.

Data Analysis:

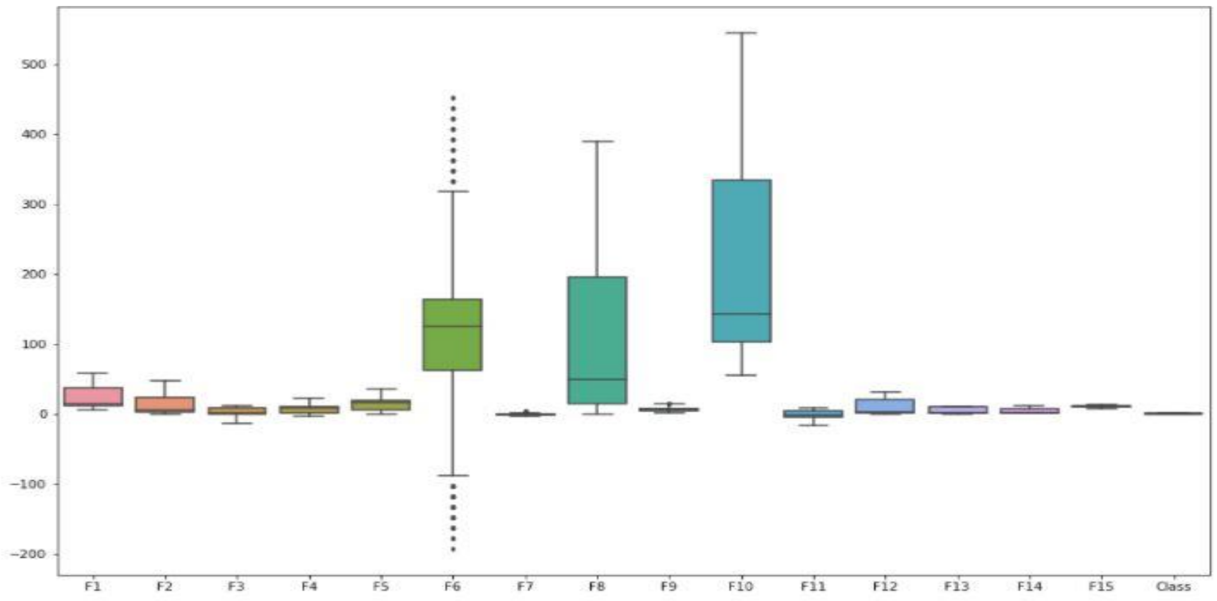
The data client provided has information about 1500 customers and each observation has 15 numerical features along with labels whether the claim is made or not for that trip. The below figures show the summary of data and missing values in 'F15' column of the data.



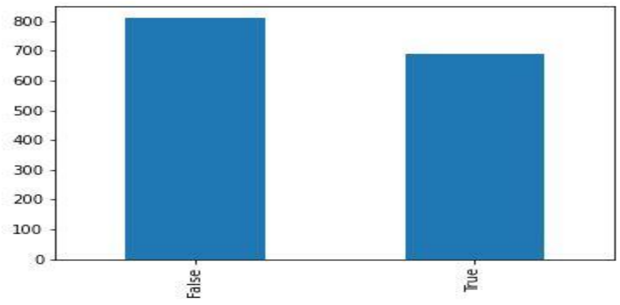
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1500 entries, 0 to 1499
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   F1      1500 non-null   float64
 1   F2      1500 non-null   float64
 2   F3      1500 non-null   float64
 3   F4      1500 non-null   float64
 4   F5      1500 non-null   float64
 5   F6      1500 non-null   float64
 6   F7      1500 non-null   float64
 7   F8      1500 non-null   int64
 8   F9      1500 non-null   float64
 9   F10     1500 non-null   float64
 10  F11     1500 non-null   float64
 11  F12     1500 non-null   float64
 12  F13     1500 non-null   int64
 13  F14     1500 non-null   float64
 14  F15     750 non-null    float64
 15  Class   1500 non-null   bool
dtypes: bool(1), float64(13), int64(2)
memory usage: 177.4 KB
```

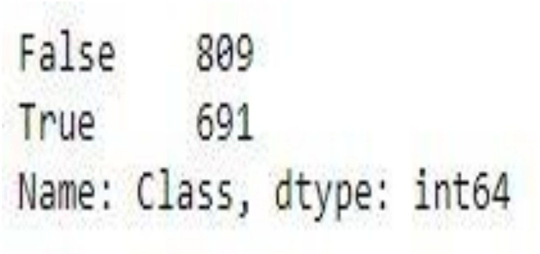


The boxplots below shows the distribution of data across all columns. The data is skewed and there are outliers in 'F6' column. Data scaling and replacing missing values is done using pipelines during model building.



The bar plot for Class column shows that 56% of data is in False class and 46% of data is in True class. This implies dataset is moderately balanced.





```
False    809
True     691
Name: Class, dtype: int64
```

The following steps are performed before building the models:

- The independent variables are separately stored in to a variable called 'X' and dependent variable 'Class' is stored in a variable called 'y'.
- Then, 80% of data is split into training set for training the model and 20% data into validation set to evaluate model performance on unseen data.
- A pipeline is created for data preprocessing. It is called preprocessor. This pipeline has 2 functions, it imputes missing values in the columns using 'SimpleImputer' library with that column mean and also performs data scaling using 'RobustScaler'. RobustScaler is chosen as there are outliers in the data.
- StratifiedKfold cross validator with 10 splits is used for cross validation and assigned to a variable called 'cv'. This cross valiadtor and the preprocessor pipeline is used in all the models.

The figure below shows the pipeline and cross validator used in model.

```
preprocesssor = Pipeline(steps =
    [
        ('imputer', SimpleImputer(strategy='mean')), # Imputation transformer for completing missing values
        ('scaler', RobustScaler()) # RoustScaler removes the median and scales the data according to the quantile range(
    ])

## Step2 : The following Crossvalidation method will be used in all the models

cv = StratifiedKFold(n_splits=10)
```

A function called ‘gridsearch’ is defined, which takes a classifier, parameters for grid search and cross validator as arguments. The estimator to grid search is a pipeline that performs data pre-processing using pre-processor pipeline created and fits the data using given machine learning algorithms.

```
# function to perform gridsearch

def gridsearch(classifier, grid_params, cv):

    """This function takes a classifier, parameters for grid search and cross validator as arguments
    and returns the classifier with high accuracy"""

    ## estimator to gridsearch is a pipeline that performs data preprocesssing and applies a fit with the given classifier.
    model = GridSearchCV(
                        estimator  = Pipeline(steps =[('preprocess',preprocesssor),('classifier',classifier)]),
                        param_grid = grid_params,  # perform gridsearch on grid parameters given
                        scoring    = 'accuracy',   # accuracy is used as scoring metric
                        n_jobs     = -1,           # n_jobs = -1,uses all the processors
                        cv         = cv,
                        verbose    = 2
                    )
    model.fit(X_train,y_train) # Fitting the training data.
    return model
```

The following machine learning algorithms are used to build the models:

1.DecisionTree:

Decision trees are easy to implement but they can create over-complex trees that do not generalize well. Hyperparameters such as maximum depth and minimum number of samples required at leaf node are tuned using GrdiSearchCV method form Sklearn library to avoid overfitting.

```
grid_parameters = {
                    'classifier__criterion':          ['gini','entropy'],
                    'classifier__max_depth':          range(2,20,1),
                    'classifier__min_samples_leaf':   range(1,10,1),
                    'classifier__min_samples_split':  range(2,10,1),
                    'classifier__splitter'          : ['best','random']
                  }
```

```
Best parameters :  {'classifier__criterion': 'entropy', 'classifier__max_depth': 18, 'classifier__min_samples_leaf': 1, 'classi
fier__min_samples_split': 7, 'classifier__splitter': 'random'}

Training Accuracy :    0.8116666666666668
Validation Accuracy :  0.7533333333333333
```

2. KNeighborsClassifier:

Choice of algorithm, distance metric, number of nearest neighbors to choose and weights to assign to data points for choosing neighbors are the hyperparameters tuned for KNN model. It is important to tune these parameters as KNN takes more time during prediction phase.

```
grid_parameters = {

                  'classifier__algorithm'  :   ['ball_tree', 'kd_tree', 'brute'],
                  'classifier__leaf_size'  :   range(1,50),
                  'classifier__n_neighbors':   range(3,20,2), # odd numbers
                  'classifier__p'          :   [1,2],
                  'classifier__weights'    :   ['uniform', 'distance'] }

# Calling the function gridsearch with KNeighborsClassifier,grid_parameters and Cross validator,saving the model in a variable.
model = gridsearch(KNeighborsClassifier(),grid_parameters,cv)
```

```
Best parameters :  {'classifier__algorithm': 'ball_tree', 'classifier__leaf_size': 1, 'classifier__n_neighbors': 17, 'classifie
r__p': 1, 'classifier__weights': 'distance'}

Training Accuracy :    0.7908333333333333
Validation Accuracy :  0.7733333333333333
```

KNN performed better than decision tree with a validation accuracy of 77%.

3. SupportVectorClassifier (SVC):

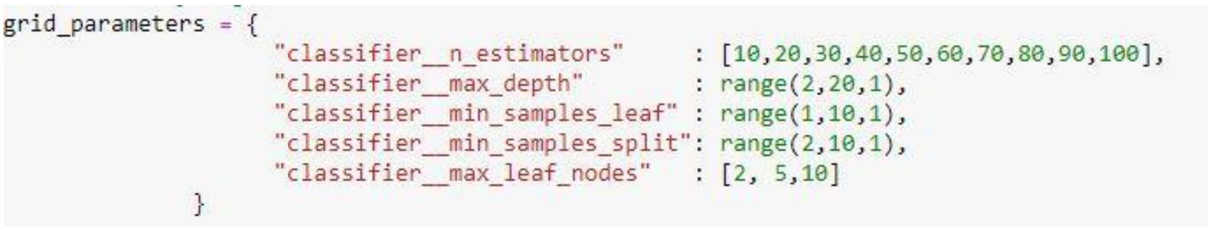
SVC has free parameters like regularization parameter ‘C’, kernel co-efficient ‘gamma’ and kernel. Kernel functions like ’rbf’ can separate data by transforming them into high dimensional space, which may not be separable using linear kernel.

```
grid_parameters = {
                   'classifier__C'      :       np.logspace(-1,2,20),
                   'classifier__gamma'  :       np.logspace(-5,5,10),
                   'classifier__kernel':        ['rbf','linear']
                  }
```

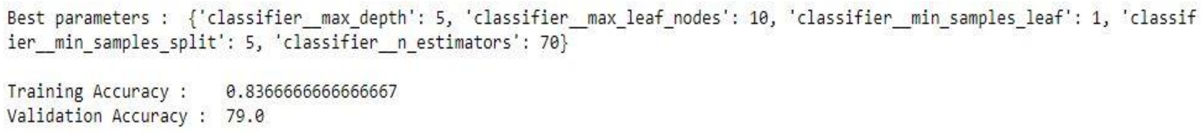
```
Best parameters :  {'classifier__C': 33.59818286283781, 'classifier__gamma': 1e-05, 'classifier__kernel': 'linear'}

Training Accuracy :    0.8833333333333334
Validation Accuracy :  0.8566666666666667

auc :  0.8590113007068569

Confusion Matrix:
 [[140  27]
 [ 16 117]]

Classification Report :
               precision    recall  f1-score   support

       False       0.90      0.84      0.87       167
        True       0.81      0.88      0.84       133

    accuracy                           0.86       300
   macro avg       0.85      0.86      0.86       300
weighted avg       0.86      0.86      0.86       300
```

Linear kernel is chosen as best parameters, which indicates that data is linearly separable.

4. RandomForestClassifier: A random forest fits a number of decision trees on various sub-samples of data and uses averaging to improve the accuracy of prediction and controls over-fitting, which often happens with decisiontrees. The number of estimators is the number of decision trees to be build, remaining parameters are same as decisiontree.



```
grid_parameters = {
                    "classifier__n_estimators"     : [10,20,30,40,50,60,70,80,90,100],
                    "classifier__max_depth"        : range(2,20,1),
                    "classifier__min_samples_leaf" : range(1,10,1),
                    "classifier__min_samples_split": range(2,10,1),
                    "classifier__max_leaf_nodes"   : [2, 5,10]
              }
```

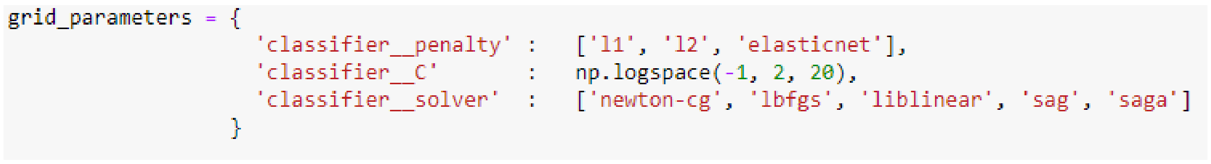


```
Best parameters :  {'classifier__max_depth': 5, 'classifier__max_leaf_nodes': 10, 'classifier__min_samples_leaf': 1, 'classif
ier__min_samples_split': 5, 'classifier__n_estimators': 70}

Training Accuracy :    0.8366666666666667
Validation Accuracy :  79.0
```

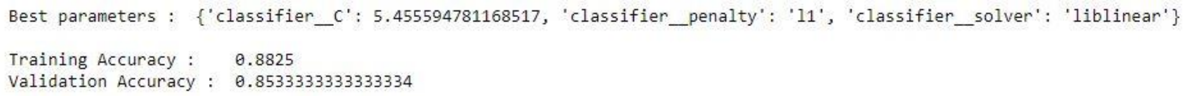
Random forest model achieved more accuracy than decision tree by reducing overfitting on training data.

5. LogisticRegression:

It is a binary classification algorithm that uses a logistic function to calculates the probability of a data belonging any one of the class. We can avoid overfitting of data using regularization.



```
grid_parameters = {
                    'classifier__penalty' :   ['l1', 'l2', 'elasticnet'],
                    'classifier__C'       :   np.logspace(-1, 2, 20),
                    'classifier__solver'  :   ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
                  }
```



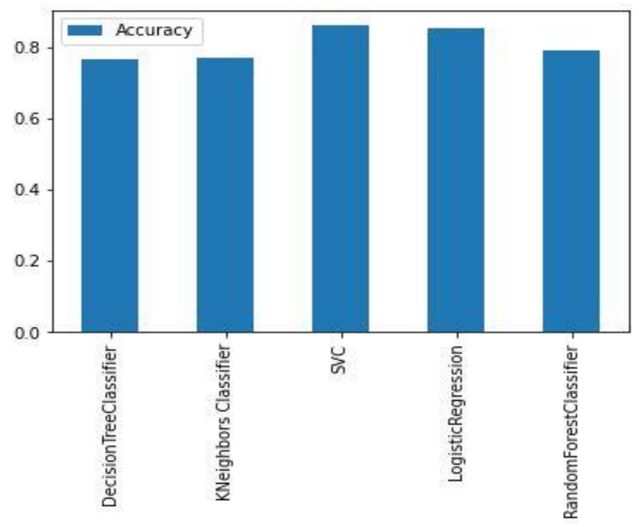
```
Best parameters :  {'classifier__C': 5.455594781168517, 'classifier__penalty': 'l1', 'classifier__solver': 'liblinear'}

Training Accuracy :    0.8825
Validation Accuracy :  0.8533333333333334
```

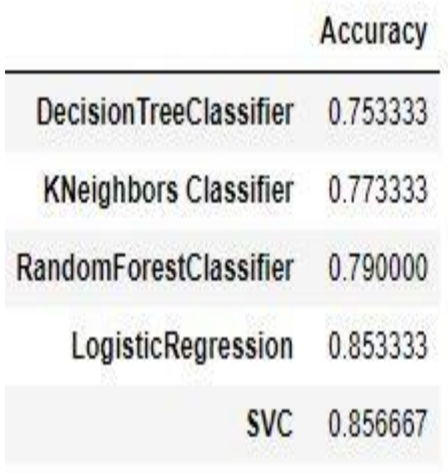
L1 is chosen as best parameter which means that the model complexity is reduced by choosing features that are important and assigning zero weights to non-relevant features. This model is achieved similar accuracy that of SVC Model.

<u>Bar graphs showing comparion of models built</u>

SVC and Logistic regression showed high accuracy of 85%, which implies that can be separated into two classes using a hyperplane.

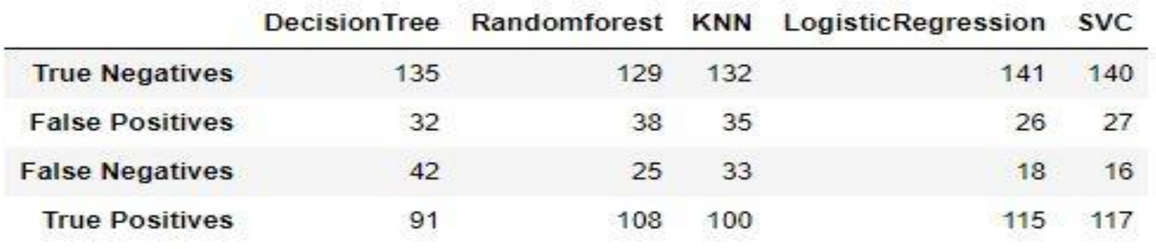
SVC made 27 mistakes while classifying positive class and 16 mistakes in classifying negative class samples. Precision, recall, AUC values are higher for SVC model hence this is chosen to predict the labels for the test data.



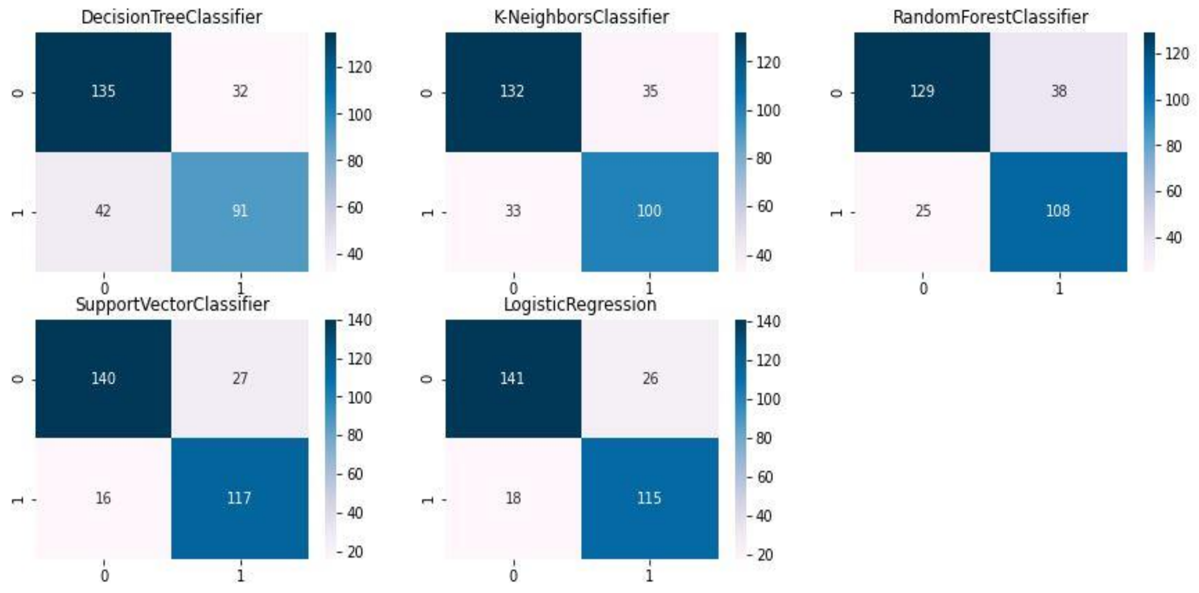


	Accuracy
DecisionTreeClassifier	0.753333
KNeighbors Classifier	0.773333
RandomForestClassifier	0.790000
LogisticRegression	0.853333
SVC	0.856667

Confusion Matrices of all the models created



	DecisionTree	Randomforest	KNN	LogisticRegression	SVC
True Negatives	135	129	132	141	140
False Positives	32	38	35	26	27
False Negatives	42	25	33	18	16
True Positives	91	108	100	115	117

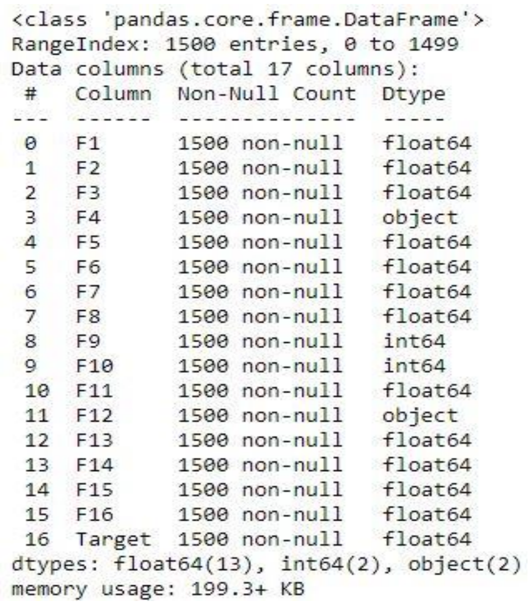


Part3 (Regression)

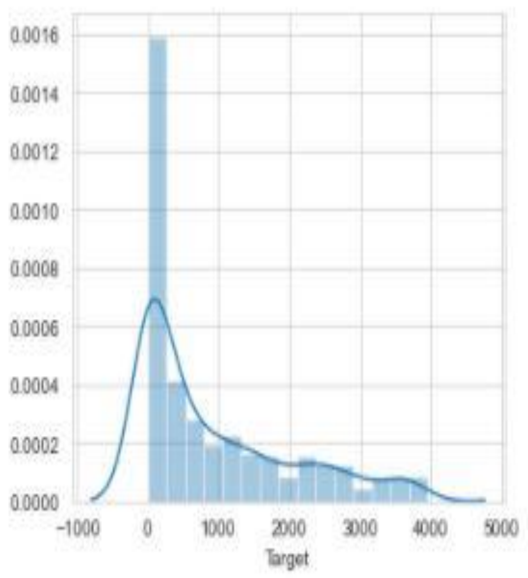
The task in part 3 of assignment involves predicting the value of claim based on data provided by the company, data includes various features about each customer and the amount they claimed.

Data Analysis:

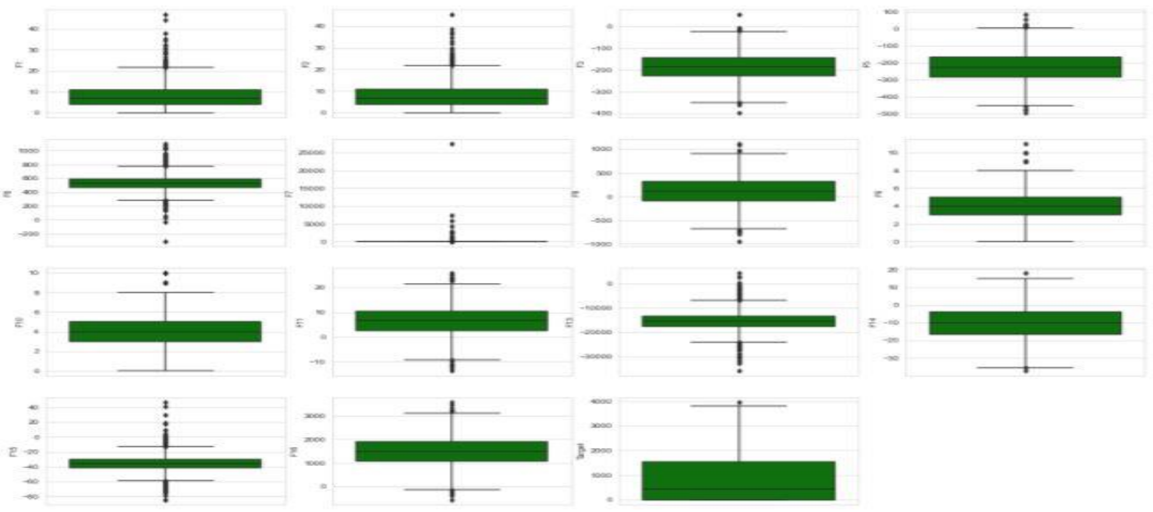
The data client provided has 17columns. The first 15 columns are independent variables and 16th column is the Target, which is the claim value. The below Figure below shows the summary of data. The dataset has no null values but there are 2 features, which are categorical.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1500 entries, 0 to 1499
Data columns (total 17 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   F1      1500 non-null   float64
 1   F2      1500 non-null   float64
 2   F3      1500 non-null   float64
 3   F4      1500 non-null   object
 4   F5      1500 non-null   float64
 5   F6      1500 non-null   float64
 6   F7      1500 non-null   float64
 7   F8      1500 non-null   float64
 8   F9      1500 non-null   int64
 9   F10     1500 non-null   int64
 10  F11     1500 non-null   float64
 11  F12     1500 non-null   object
 12  F13     1500 non-null   float64
 13  F14     1500 non-null   float64
 14  F15     1500 non-null   float64
 15  F16     1500 non-null   float64
 16  Target  1500 non-null   float64
dtypes: float64(13), int64(2), object(2)
memory usage: 199.3+ KB
```



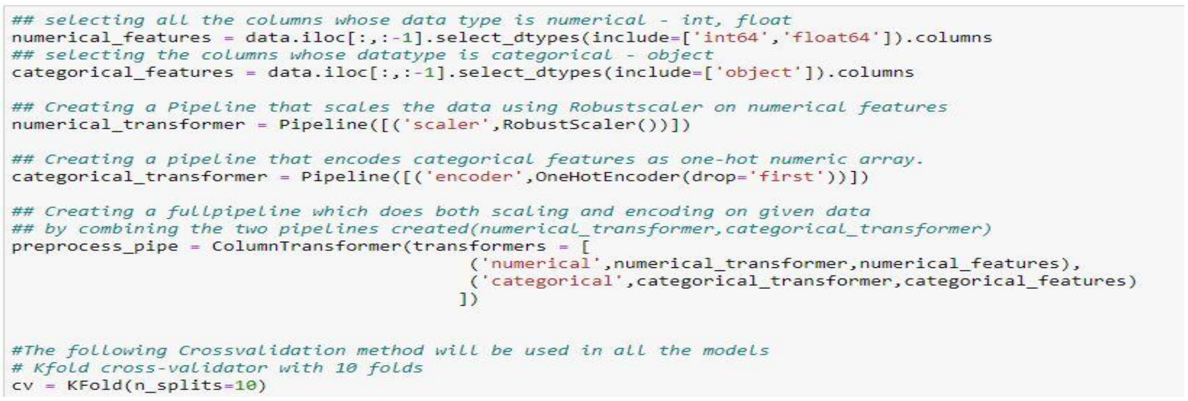
Distribution plot for Target variable shows skewness towards right, with a peak at zero. This skewness can adversely effect the model's performance especially models built using regression algorithms by acting as an outlier. The figure below shows the data distribution in each column, there are outliers in most of the columns, to deal with the outliers **RobustScaler** is used to scale the data.



The following steps are performed before building the models:

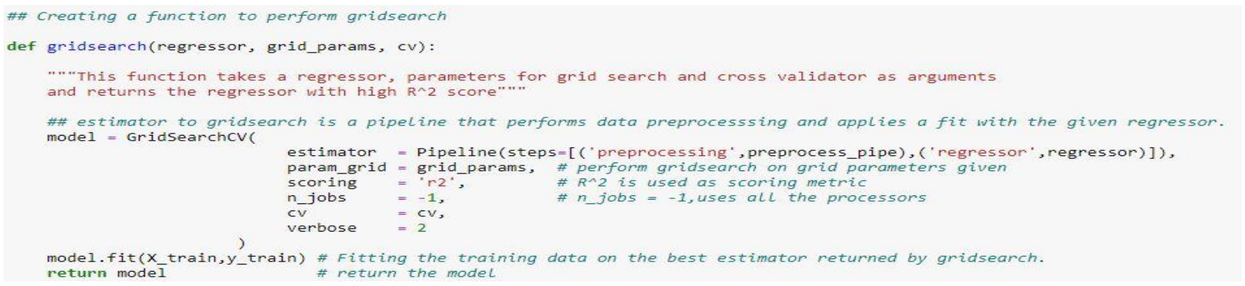
- First the independent variables are separately stored in to a variable called 'X' and dependent variable 'Target' is stored in a variable called 'y'.

- Then, 80% of data is split into training set for training the model and 20% data into validation set to evaluate model performance on unseen data.
- A pipeline is created for data preprocessing. It is called preprocess_pipe. This pipeline has 2 functions, it performs data scaling using 'RobustScaler' and encodes categorical varaibles using 'OneHotEncoder'.
- Kfold cross validator with 10 splits is used for cross validation and assigned to a variable called 'cv'. This cross valiadtor and the preprocess_pipe pipeline is used in all the models.



```
## selecting all the columns whose data type is numerical - int, float
numerical_features = data.iloc[:,:-1].select_dtypes(include=['int64','float64']).columns
## selecting the columns whose datatype is categorical - object
categorical_features = data.iloc[:,:-1].select_dtypes(include=['object']).columns

## Creating a Pipeline that scales the data using Robustscaler on numerical features
numerical_transformer = Pipeline([('scaler',RobustScaler())])

## Creating a pipeline that encodes categorical features as one-hot numeric array.
categorical_transformer = Pipeline([('encoder',OneHotEncoder(drop='first'))])

## Creating a fullpipeline which does both scaling and encoding on given data
## by combining the two pipelines created(numerical_transformer,categorical_transformer)
preprocess_pipe = ColumnTransformer(transformers = [
                                        ('numerical',numerical_transformer,numerical_features),
                                        ('categorical',categorical_transformer,categorical_features)
                                       ])


#The following Crossvalidation method will be used in all the models
# Kfold cross-validator with 10 folds
cv = KFold(n_splits=10)
```

A function called 'gridsearch' is defined, which takes a regressor, parameters for grid search and cross validator as arguments. The estimator to grid search is a pipeline that performs data pre-processing using preprocess_pipe pipeline created and fits the data using given machine learning algorithm.

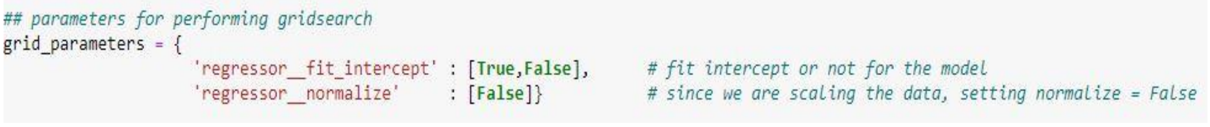


```
## Creating a function to perform gridsearch

def gridsearch(regressor, grid_params, cv):

    """This function takes a regressor, parameters for grid search and cross validator as arguments
    and returns the regressor with high R^2 score"""

    ## estimator to gridsearch is a pipeline that performs data preprocesssing and applies a fit with the given regressor.
    model = GridSearchCV(
                            estimator   = Pipeline(steps=[('preprocessing',preprocess_pipe),('regressor',regressor)]),
                            param_grid  = grid_params,   # perform gridsearch on grid parameters given
                            scoring     = 'r2',          # R^2 is used as scoring metric
                            n_jobs      = -1,            # n_jobs = -1,uses all the processors
                            cv          = cv,
                            verbose     = 2
                        )
    model.fit(X_train,y_train) # Fitting the training data on the best estimator returned by gridsearch.
    return model               # return the model
```

The following machine learning algorithms are used to build the models:

1.Linear Regression:

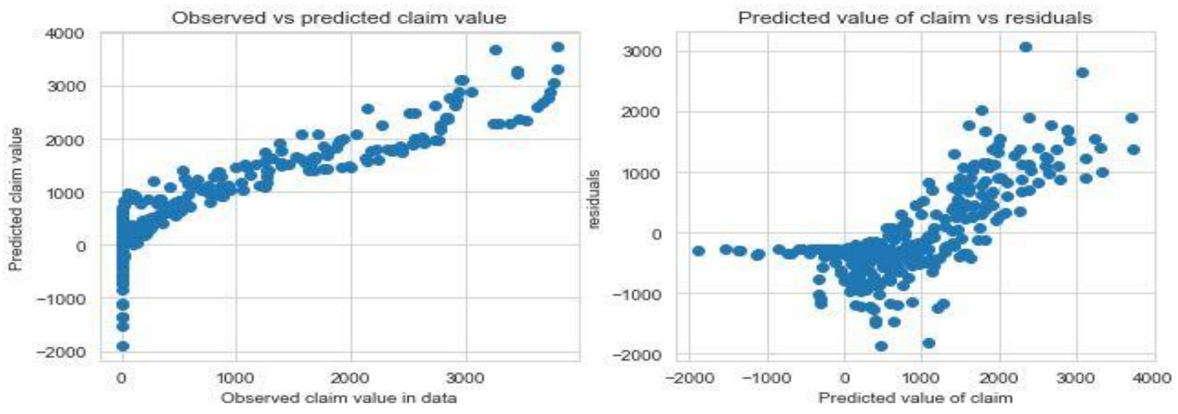
There are only 2 free parameters to tune in Linear Regression fit intercept and normalize, as the scaling is performed during pre-processing, setting it to False and allowing model to choose whether to fit intercept or not.



```
## parameters for performing gridsearch
grid_parameters = {
                    'regressor__fit_intercept' : [True,False],      # fit intercept or not for the model
                    'regressor__normalize'     : [False]}           # since we are scaling the data, setting normalize = False
```

These are best parameters and scores obtained:

```
Best Parameters : {'regressor__fit_intercept': True, 'regressor__normalize': False}
Training R^2    : 0.7836010446586683
Validation R^2  : 0.818413506670992
RMSE            : 459.2209483019344
```



As anticipated, due to skewed target variable, the model built using LinearRegression did not perform well during prediction, which can be seen in scatterplot of observed vs predicted value. One of the assumptions of linear regression is that residuals have constant variance at every levels of X. The scatterplot of predicted value vs residuals shows the clear violation of this. This violation increases the variance of the regression coefficient estimates, but the regression model fails to pick this, as result making it to declare a feature as statistically significant, when actually it is not. Therefore, predictions made by fitting these co-efficients yields poor results.

2. SupportVectorRegressor:

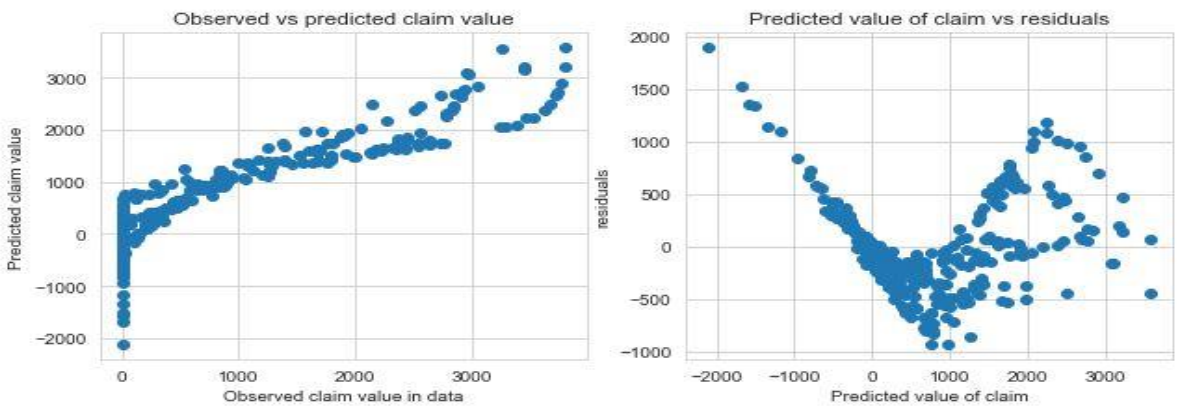
The free parameters in SVR includes epsilon, C, gamma and kernel functions.

```
grid_parameters = {
                     'regressor__C'       : np.logspace(-1,2,25),
                     'regressor__epsilon' : [0.01,0.05,0.1,0.2,0.3,0.4,0.5],
                     'regressor__gamma'   : [1,0.1,0.01,0.001,0.0001],
                     'regressor__kernel'  : ['linear', 'rbf']
             }
```

Along with linear kernel, a non-linear kernel 'rbf' is used to see if tranformation in high dimensional space produces any better results than using just linear kernel. Using Epsilon felxibility is provided to model to not to penalise values that are epsilon distant from actual value.

Following are the best parameters and scores obtained using SVR model:

```
Best Parameters : {'regressor__C': 74.98942093324558, 'regressor__epsilon': 0.5, 'regressor__gamma': 1, 'regressor__kernel': 'l
inear'}

Training R^2    : 0.7678089692641399

Validation R^2  : 0.8093100715492929

RMSE            : 470.59119604255517
```



SVR also not able to perform well because of the skewness in target. For zero and values above 300, the we can see that model predictions are terrible. The model performance is similar to that of linear regression.

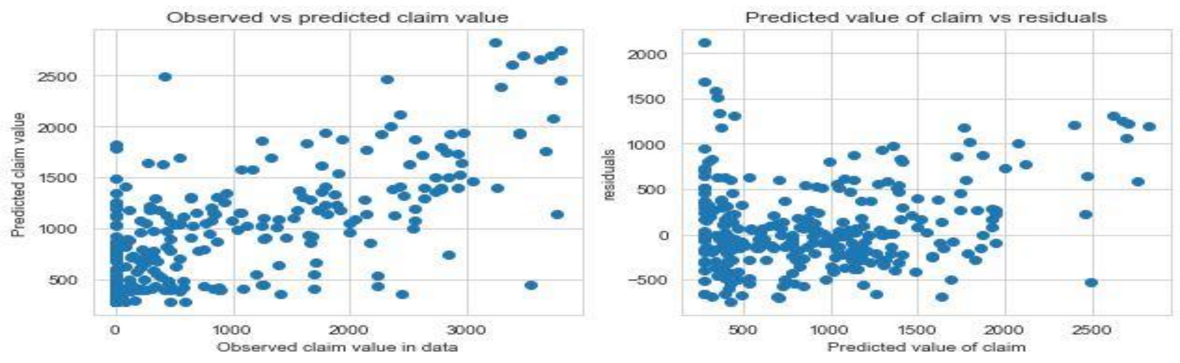
3. RandomForestRegressor:

Model is given following set of parameters to choose for finding the best estimator.

```
grid_parameters = {
                    "regressor__n_estimators":    [10,20,50,70,100],
                    "regressor__max_depth":        range(1,30,1),
                    "regressor__min_samples_leaf": range(1,10,1),
                    "regressor__max_leaf_nodes":   [2, 5,10]
            }
```

Following are the best parameters and scores obtained:

```
Best Parameters : {'regressor__max_depth': 24, 'regressor__max_leaf_nodes': 10, 'regressor__min_samples_leaf': 6, 'regressor__
_estimators': 50}

Training R^2    : 0.5872103902061914

Validation R^2  : 0.463634618658075

RMSE            : 789.2421842073259
```



Randomforest is only able to capture 58% variance in data, as a result it didn’t generalize well on validation data and the score is 0.48.

4. MLP Regressor:

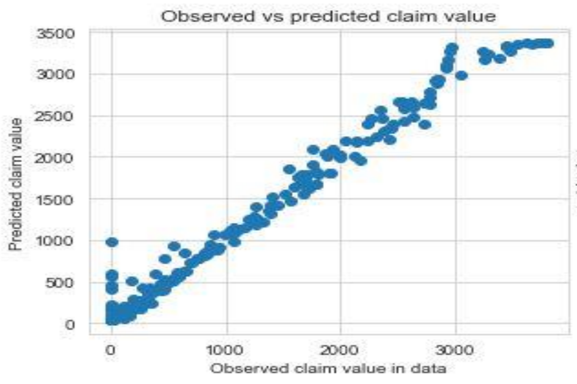
MLP regressor impliments multi-layer perceptron that optimizes squared-loss using stochastic gradient descent back propogation algorithm.

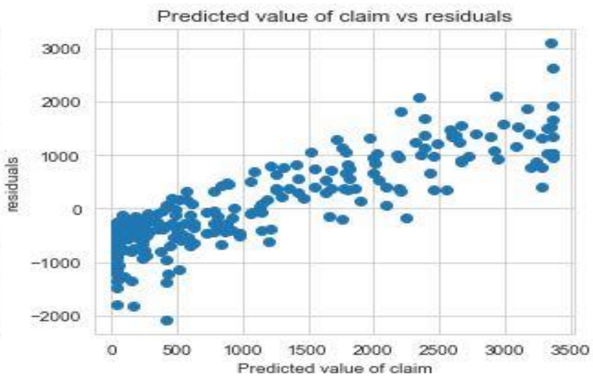
```
grid_parameters = {
    'regressor__hidden_layer_sizes':   np.arange(25,100,15),
    'regressor__activation' :          ['relu','logistic'],
    'regressor__learning_rate' :       ['constant', 'adaptive'],
    'regressor__learning_rate_init' :  [0.1,0.2,0.3, 0.4,0.5,0.6]

}

# Calling the function gridsearch with RandomForestRegressor,grid_parameters and Cross validator,saving the model in a variable.
model = gridsearch(MLPRegressor(random_state=1,early_stopping=True,max_iter=5000),grid_parameters,cv)
```

The model trains iteratively by updating parameters at each step by computing partial derivatives with respect to model parameters.

These are the best parametersand scores obtained:

```
Best Parameters : {'regressor__activation': 'logistic', 'regressor__hidden_layer_sizes': 70, 'regressor__learning_rate': 'const
ant', 'regressor__learning_rate_init': 0.1}

Training R^2    : 0.9797998872579452

Validation R^2  : 0.9843746942655459

RMSE            : 134.7082462040153
```

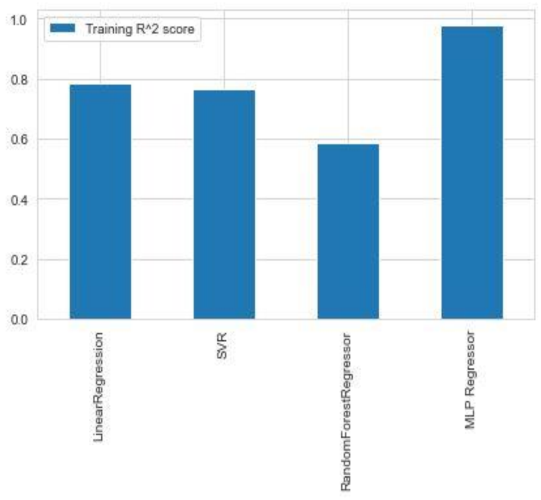


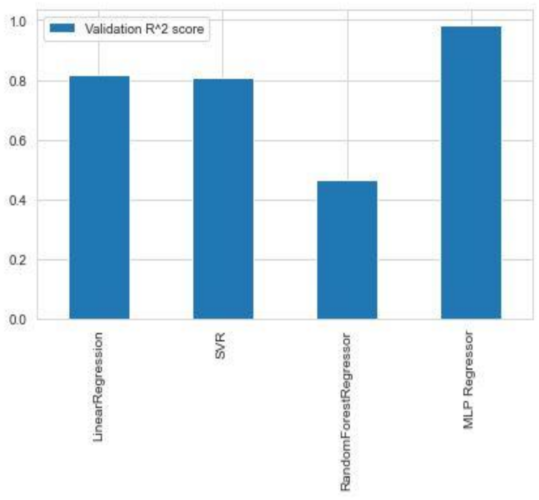


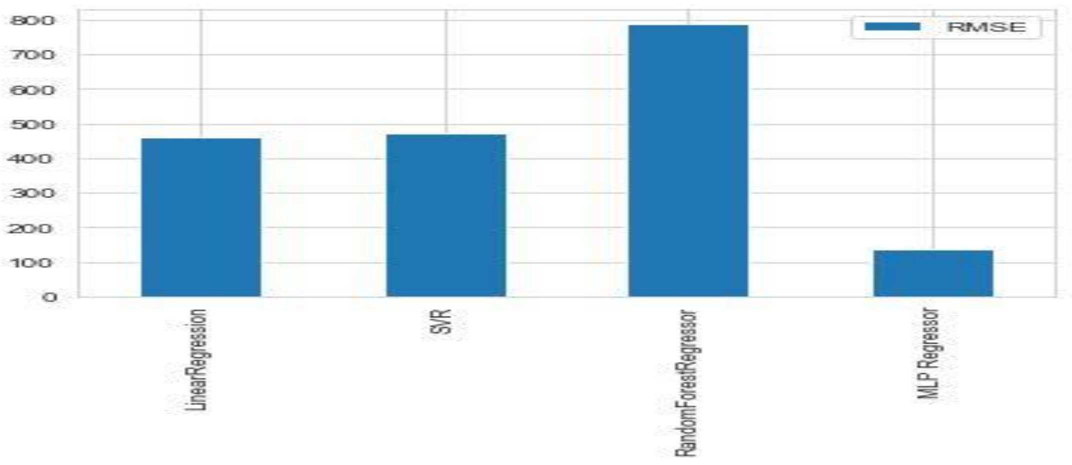
Using MLP, model is able to capture the general trend in data and made just 2% error during prediction.

<u>Bar graphs showing comparion of models built</u>

Both during training and validation MLP outperformed remaining models with 0.97 training and 0.98 validation score. The RMSE of the model is only 134.7. MLP Regressor is used to predict claim value in test data.


Training R^2 score
1.0
0.8
0.6
0.4
0.2
0.0
LinearRegression
SVR
RandomForestRegressor
MLP Regressor


Validation R^2 score
1.0
0.8
0.6
0.4
0.2
0.0
LinearRegression
SVR
RandomForestRegressor
MLP Regressor


RMSE
800
700
600
500
400
300
200
100
0
LinearRegression
SVR
RandomForestRegressor
MLP Regressor