Import libraries and dataset.

```
##import basic libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statistics
import random
from copy import deepcopy, copy
import itertools
from collections import Counter
import math
from graphviz import Digraph, Source, Graph
import scipy
from sklearn.metrics import pairwise_distances
from scipy.stats import multivariate_normal as mvn
```

```
##import Data from local disk.
##data is from final project discussion board: https://osf.io/wx7ck
from google.colab import files
uploaded = files.upload()
```

Choose Files DatasetML.csv

- **DatasetML.csv**(text/csv) - 36071309 bytes, last modified: 12/13/2020 - 100% done

```
Saving DatasetML.csv to DatasetML.csv
```

```
##import data to pandas
data = pd.read_csv('DatasetML.csv', encoding = "ISO-8859-1")
data.head()
```

twarm	citizenship	d_donotuse	diseaseframinga	diseaseframingb	ethnicity	feedb
58	US	0.264358628	200 people will be saved		2	Your sugge s prefere for Ma
95	US	-0.14431008	200 people will be saved		2	Your sug little c prefere f
				1/3 probability		Your sugge

sorting columns and clean dataset

for

```
##sorting out the columns
halfhalf_answer = ['flagsupplement1', 'flagsupplement2', 'flagsupplement3', 'iatexplic
                'iatexplicitart3', 'iatexplicitart4', 'iatexplicitart5', 'iatexplicita
                'iatexplicitmath2', 'iatexplicitmath3', 'iatexplicitmath4', 'iatexplic
                'sysjust1', 'sysjust2', 'sysjust3', 'sysjust4', 'sysjust5', 'sysjust6'
                'mturk.non.US', 'exprace']

number_answer = [  'anchoring1bkm', 'priorexposure8',
                    'mturk.duplicate', 'mturk.exclude.null', 'mturk.keep', 'mturk.excl
                    'gamblerfallacyb', 'numparticipants_actual', 'session_id', 'priore
                    'priorexposure5', 'priorexposure6', 'priorexposure7',
                    'priorexposure9', 'priorexposure10', 'priorexposure11', 'priorexpo
                    'numparticipants', 'age', 'anchoring1a', 'priorexposure1', 'priore
                    'anchoring1b', 'anchoring2a', 'anchoring2b', 'anchoring3a', 'ancho
                    'anchoring4b', 'artwarm', 'd_donotuse', 'ethnicity', 'flagdv1', 'f
                    'flagdv4', 'flagdv5', 'flagdv6', 'flagdv7', 'flagdv8', 'gamblerfal
                    'imaginedexplicit1', 'imaginedexplicit2', 'imaginedexplicit3', 'im
                    'major', 'mathwarm', 'moneyagea', 'moneyageb', 'moneygendera', 'mo
                    'omdimc3trt', 'quotea', 'quoteb', 'sunkcosta', 'sunkcostb', 'user_
                    'order', 'meanlatency', 'meanerror', 'block2_meanerror', 'block3_m
                    'block6_meanerror', 'lat11', 'lat12', 'lat21', 'lat22', 'sd1', 'sc
                    'flagdv1',  'flagdv2',  'flagdv3','flagdv4',  'flagdv5', 'flagdv6'
                    'flagdv8','scalesorder', 'reciprocorder', 'diseaseforder', 'quotec
                    'sunkcostorder', 'anchorinorder', 'allowedforder', 'gamblerforder'
                    'iatorder']


non_string_answer = halfhalf_answer + number_answer
string_answer = sorted([col for col in data.columns if col not in non_string_answer])


data = data.replace({'': np.nan, ' ': np.nan, '.': np.nan})##replace null elements to
```

```
for col in string_answer:
    data[col] = data[col].astype(str)


for col in number_answer:
    data[col] = pd.to_numeric(data[col],errors='coerce')


data["flagsupplement1"] = data["flagsupplement1"].apply(lambda x:
                                                        '10' if x == 'Very much' else
                                                        ('1' if x == 'Not at all' else x))
data["flagsupplement2"] = data["flagsupplement2"].apply(lambda x:
                                                        '1' if x == 'Democrat' else
                                                        ('7' if x == 'Republican' else x))
data["flagsupplement3"] = data["flagsupplement3"].apply(lambda x:
                                                        '1' if x == 'Liberal' else
                                                        ('7' if x == 'Conservative' else x

data["iatexplicitart1"] = data["iatexplicitart1"].apply(lambda x:
                                                        '1' if x == 'Very bad' else
                                                        ('2' if x == 'Moderately bad' else
                                                        ('3' if x == 'Slightly bad' else
                                                        ('4' if x == 'Neither good nor bad
                                                        ('5' if x == 'Slightly good' else
                                                        ('6' if x == 'Moderately good' els
                                                        ('7' if x == 'Very good' else x))))

data["iatexplicitart2"] = data["iatexplicitart2"].apply(lambda x:
                                                        '1' if x == 'Very sad' else
                                                        ('2' if x == 'Moderately sad' else
                                                        ('3' if x == 'Slightly sad' else
                                                        ('4' if x == 'Neither happy Nor Sa
                                                        ('5' if x == 'Slightly happy' or '
                                                        ('6' if x == 'Moderately happy' el
                                                        ('7' if x == 'Very happy' else x))
data["iatexplicitart3"] = data["iatexplicitart3"].apply(lambda x:
                                                        '1' if x == 'Very Ugly' else
                                                        ('2' if x == 'Moderately Ugly' els
                                                        ('3' if x == 'Slightly Ugly' else
                                                        ('4' if x == 'Neither Beautiful Nc
                                                        ('5' if x == 'Slightly Beautiful'
                                                        ('6' if x == 'Moderately Beautiful
                                                        ('7' if x == 'Very Beautiful' else
data["iatexplicitart4"] = data["iatexplicitart4"].apply(lambda x:
                                                        '1' if x == 'Very Disgusting' else
                                                        ('2' if x == 'Moderately Disgustin
                                                        ('3' if x == 'Slightly Disgusting'
                                                        ('4' if x == 'Neither Delightful N
                                                        ('5' if x == 'Slightly Delightful'
                                                        ('6' if x == 'Moderately Delightfu
                                                        ('7' if x == 'Very Delightful' els
```

```
data["iatexplicitart5"] = data["iatexplicitart5"].apply(lambda x:
                                                        '1' if x == 'Very Avoid' else
                                                        ('2' if x == 'Moderately Avoid' el
                                                        ('3' if x == 'Slightly Avoid' else
                                                        ('4' if x == 'Neither Approach Nor
                                                        ('5' if x == 'Slightly Approach' e
                                                        ('6' if x == 'Moderately Approach'
                                                        ('7' if x == 'Very Approach' else
data["iatexplicitart6"] = data["iatexplicitart6"].apply(lambda x:
                                                        '1' if x == 'Very Afraid' else
                                                        ('2' if x == 'Moderately Afraid' e
                                                        ('3' if x == 'Slightly Afraid' els
                                                        ('4' if x == 'Neither Unafraid Nor
                                                        ('5' if x == 'Slightly Unafraid' e
                                                        ('6' if x == 'Moderately Unafraid'
                                                        ('7' if x == 'Very Unafraid' else

data["iatexplicitmath1"] = data["iatexplicitmath1"].apply(lambda x:
                                                        '1' if x == 'Very bad' else
                                                        ('2' if x == 'Moderately bad' else
                                                        ('3' if x == 'Slightly bad' else
                                                        ('4' if x == 'Neither good nor bad
                                                        ('5' if x == 'Slightly good' else
                                                        ('6' if x == 'Moderately good' els
                                                        ('7' if x == 'Very good' else x))
data["iatexplicitmath2"] = data["iatexplicitmath2"].apply(lambda x:
                                                        '1' if x == 'Very sad' else
                                                        ('2' if x == 'Moderately sad' else
                                                        ('3' if x == 'Slightly sad' else
                                                        ('4' if x == 'Neither happy Nor Sa
                                                        ('5' if x == 'Slightly happy' or '
                                                        ('6' if x == 'Moderately happy' el
                                                        ('7' if x == 'Very happy' else x))
data["iatexplicitmath3"] = data["iatexplicitmath3"].apply(lambda x:
                                                        '1' if x == 'Very Ugly' else
                                                        ('2' if x == 'Moderately Ugly' els
                                                        ('3' if x == 'Slightly Ugly' else
                                                        ('4' if x == 'Neither Beautiful No
                                                        ('5' if x == 'Slightly Beautiful'
                                                        ('6' if x == 'Moderately Beautiful
                                                        ('7' if x == 'Very Beautiful' else
data["iatexplicitmath4"] = data["iatexplicitmath4"].apply(lambda x:
                                                        '1' if x == 'Very Disgusting' else
                                                        ('2' if x == 'Moderately Disgustin
                                                        ('3' if x == 'Slightly Disgusting'
                                                        ('4' if x == 'Neither Delightful N
                                                        ('5' if x == 'Slightly Delightful'
                                                        ('6' if x == 'Moderately Delightfu
                                                        ('7' if x == 'Very Delightful' els
data["iatexplicitmath5"] = data["iatexplicitmath5"].apply(lambda x:
                                                        '1' if x == 'Very Avoid' else
                                                        ('2' if x == 'Moderately Avoid' el
```

```
                                                    ('3' if x == 'Slightly Avoid' else
                                                    ('4' if x == 'Neither Approach Noı
                                                    ('5' if x == 'Slightly Approach' e
                                                    ('6' if x == 'Moderately Approach'
                                                    ('7' if x == 'Very Approach' else
data["iatexplicitmath6"] = data["iatexplicitmath6"].apply(lambda x:
                                                    '1' if x == 'Very Afraid' else
                                                    ('2' if x == 'Moderately Afraid' e
                                                    ('3' if x == 'Slightly Afraid' els
                                                    ('4' if x == 'Neither Unafraid Noı
                                                    ('5' if x == 'Slightly Unafraid' e
                                                    ('6' if x == 'Moderately Unafraid'
                                                    ('7' if x == 'Very Unafraid' else

for col in halfhalf_answer:
    if "sysjust" in col:
        data[col] = data[col].apply(lambda x:
                                    '1' if x == 'Strongly disagree' else
                                    ('7' if x == 'Strongly agree' else x))

data["mturk.non.US"] = data["mturk.non.US"].apply(lambda x: '1' if x == 'non-US IP adc
data["exprace"] = data["exprace"].apply(lambda x: '11' if x == 'brazilwhite' else
                                        ('12' if x == 'brazilblack' else
                                        ('13' if x == 'brazilbrown' else
                                        ('14' if x == 'chinese' else
                                        ('15' if x == 'malay' else
                                        ('16' if x == 'dutch' else x))))))
for col in halfhalf_answer:
    data[col] = data[col].astype(str)


##clean unrelated data
date_answer = [col for col in data.columns if '_date' in col]
useless_url = [col for col in data.columns if '_url.' in col]
null_answer = ['task_status', 'task_sequence', 'beginlocaltime']
useless_answer = ["session_created_by", 'study_url','study_name','text',
             'session_id', 'user_id', "previous_session_id", "expcomments"]
unrelated_answer = date_answer + null_answer + useless_answer + useless_url + string_a
data.drop(unrelated_answer,inplace=True, axis=1,errors = "ignore")
data.shape
```

```
(6344, 112)
```

```
final_answer = number_answer + string_answer


data.head()
```

;	flagdv6	flagdv7	flagdv8	flagsupplement1	flagsupplement2	flagsupplement3	ga
)	4.0	3.0	4.0	10	4	4	
)	7.0	1.0	4.0	8	4	7	
)	7.0	5.0	2.0	10	4	4	
)	3.0	3.0	1.0	10	2	3	
)	2.0	3.0	1.0	5	3	4	

Double-click (or enter) to edit

```
class LinearRegression():
    def __init__(self, weights=None, bias=None):
        self.w = weights
        self.b = bias

    def predict(self, X):
        return (np.dot(X, self.w)) + self.b

    def fit(self, X, y, alpha=0.001, iterations=100):
        n_samples, n_features = X.shape
        self.w = np.zeros(shape=(n_features, 1))
        self.b = 0
        J = []
        y = y.reshape(-1, 1)
        for i in range(iterations):
            y_hat = self.predict(X)
            cost = (1/n_samples)*np.sum((y_hat-y)**2)
            J.append(cost)
            dJ_dw = (2/n_samples)*np.dot(X.T, (y_hat-y))
            dJ_db = (2/n_samples)*np.sum((y_hat-y))
            self.w = self.w - alpha*dJ_dw
            self.b = self.b - alpha*dJ_db

    def get_mse(self, y_true, y_pred):
        return np.mean((y_true - y_pred)**2)


class RidgeRegression():

    def __init__(self,
                 bias = None,
                 weights = None,
                 lambda_param = 10,
                 fit_intercept = True):
        self.bias = bias
        self.weights = weights
        self.fit_intercept = fit_intercept
```

```
        self.lambda_param = lambda_param

    def fit(self, X, y):
        if self.fit_intercept:
            X = np.column_stack((np.ones(len(X)), X))
        else:
            X = np.column_stack((np.zeros(len(X)), X))

        self.all_weights = np.linalg.inv(np.dot(X.T, X) + \
                             self.lambda_param * np.identity(X.shape[1])).dot(X.T).dot(
        self.weights = self.all_weights[1:]
        self.bias = self.all_weights[0]

    def predict(self, X):
        self.weights = self.weights.reshape(1, -1)
        predictions = self.bias + np.dot(self.weights, X.T)
        return predictions[0]

    def get_mse(self, y_true, y_pred):
        return np.mean((y_true - y_pred)**2)


class LassoRegression():
    def __init__(self,
                 bias = None,
                 weights = None,
                 lambda_param = 10,
                 max_iters = 100,
                 fit_intercept = True):
        self.bias = 0
        self.lambda_param = lambda_param
        self.max_iters = max_iters
        self.fit_intercept = fit_intercept

    def _soft_threshold(self, x, lambda_):
        if x > 0.0 and lambda_ < abs(x):
            return x - lambda_
        elif x < 0.0 and lambda_ < abs(x):
            return x + lambda_
        else:
            return 0.0

    def fit(self, X, y):
        if self.fit_intercept:
            X = np.column_stack((np.ones(len(X)), X))
        else:
            X = np.column_stack((np.zeros(len(X)), X))

        row_length, column_length = X.shape
        self.weights = np.zeros((1, column_length))[0]
        if self.fit_intercept:
            self.weights[0] = np.sum(y - \
```

```
                                np.dot(X[:, 1:], self.weights[1:]))/(X.shape[0])
                for iteration in range(self.max_iters):
            start = 1 if self.fit_intercept else 0

            for j in range(start, column_length):
                tmp_weights = self.weights.copy()
                tmp_weights[j] = 0.0
                r_j = y - np.dot(X, tmp_weights)
                arg1 = np.dot(X[:, j], r_j)
                arg2 = self.lambda_param * X.shape[0]

                self.weights[j] = self._soft_threshold(arg1, arg2)/(X[:, j]**2).sum()


                if self.fit_intercept:
                    self.weights[0] = np.sum(y - np.dot(X[:, 1:], self.weights[1:]))/(

        self.bias = self.weights[0]
        self.weights = self.weights[1:]

    def predict(self, X):
        self.weights = self.weights.reshape(1, -1)
        predictions = self.bias + np.dot(self.weights, X.T)
        return predictions[0]

    def get_mse(self, y_true, y_pred):
        return np.mean((y_true - y_pred)**2)


class KNeighbours():
    def __init__(self, k = 5, distance_metric = 'euclid', problem = "classify"):
        self.k = k
        self.distance_metric = distance_metric
        self.problem = problem
        self.prediction_functions = {'classify': self._top_k_votes,
                                     'regress': self._top_k_mean}
        self.eval_functions = {'classify': self._get_accuracy,
                               'regress': self._get_mse}

    def fit(self, X, y):
        self.X = np.asarray(X)
        self.y = np.asarray(y)

    def _euclidien_distance(self, x):
        return np.sqrt(np.sum((x - self.X)**2, axis = 1))

    def _top_k_mean(self, top_k):
        return np.mean(top_k)

    def _top_k_votes(self, top_k):
        return max(top_k, key = list(top_k).count)
```

```
    def _get_accuracy(self, pred, y):
        return np.mean((pred == y))

    def _get_mse(self, pred, y):
        return np.mean((pred - y)**2)

    def predict(self, X):
        preds = list()
        X = np.asarray(X)
        for x in X:
            distances = self._euclidien_distance(x)

            distances = zip(*(distances, self.y))

            distances = sorted(distances, key = lambda x: x[0])

            top_k = distances[:(self.k)]

            top_k = np.array(top_k)
            top_k = top_k[:, 1]

            pred = self.prediction_functions[self.problem](top_k)
            preds.append(pred)

        return preds

    def evaluate(self, pred, y):
        eval_func = self.eval_functions[self.problem]
        return eval_func(pred, y)


class Pipeline():
    def __init__(self, data, models, final_answer):
        self.data_orig = data
        self.data = data
        self.final_answer = final_answer
        self.data = self.data.replace({'': np.nan, ' ': np.nan, '.': np.nan, 'nan': np

        for col in self.final_answer[1]:
            if col in self.data.columns:
                self.data[col] = self.data[col].astype("category").cat.codes
                self.data[col] = self.data[col].replace({-1: np.nan})
        self.lin_reg = models[0]
        self.ridge = models[1]
        self.lasso = models[2]
        self.models2 = ["linear", "ridge", "knn"]

    def count_missing(self, data):
        return data.isnull().sum()

    def missing_value_perc(self):
        missing_value_data = (self.data.isnull().sum()*100/len(self.data)).reset_index
```

```
        missing_value_data.columns = ["feature", "perc"]
        full_value_data = missing_value_data[missing_value_data["perc"] == 0]
        missing_value_data = missing_value_data[missing_value_data["perc"] > 0]
        missing_value_data = missing_value_data.sort_values(by=['perc'])
        return missing_value_data, full_value_data


    def missing_value_update_check(self, data):
        missing_value_data = (data.isnull().sum()*100/len(data)).reset_index()
        missing_value_data.columns = ["feature", "perc"]
        full_value_data = missing_value_data[missing_value_data["perc"] == 0]
        missing_value_data = missing_value_data[missing_value_data["perc"] > 0]
        missing_value_data = missing_value_data.sort_values(by=['perc'])
        print(missing_value_data.shape, full_value_data.shape)


    def create_full_data(self, data):
        data_without_nan = data.drop(pd.isnull(data).any(1).nonzero()[0])
        return data_without_nan


    def min_max_scalar(self, data):
        d = {}
        for col in data.columns:
            if col not in d:
                d[col] = [min(data[col]), max(data[col])]

        df_min_max = pd.DataFrame.from_dict(d)
        return df_min_max

    def scale_transform(self, data):
        for col in self.df_min_max.columns:
            min_val = self.df_min_max[col][0]
            max_val = self.df_min_max[col][1]
            if max_val != min_val:
                denom = (max_val - min_val)
            else:
                denom = 0.0001
            data[col] = (data[col] - min_val)/ denom
        return data


    def split(self, max_index, n_folds=10):
        n = max_index
        idxs = np.arange(n)
        fold_sizes = (n // n_folds) * np.ones(n_folds, dtype=np.int)
        fold_sizes[:n % n_folds] += 1
        current = 0
        splits = []
        for fold_size in fold_sizes:
            start, stop = current, current + fold_size
            val = idxs[start:stop]
```

```
            splits.append(list(val))
            current = stop

        return splits


    def cross_validation(self, train_data, X_test, y_test, model, feature_name):
        preds = []
        losses = []

        max_index = len(train_data[feature_name])
        split_splits = self.split(max_index, n_folds=10)

        if model == "mean_mode":
            for i in range(len(split_splits)):
                k_copy = split_splits.copy()
                del k_copy[i]
                val = split_splits[i]
                train_i = list(itertools.chain.from_iterable(k_copy))
                val_data = train_data.iloc[val,:]
                new_train_data = train_data.iloc[train_i,:]

                X_train = new_train_data.drop(feature_name, axis=1)

                self.df_min_max = self.min_max_scalar(X_train)
                X_train = self.scale_transform(X_train)
                y_train = new_train_data[feature_name]

                X_val = val_data.drop(feature_name, axis=1)
                X_val = self.scale_transform(X_val)
                y_val = val_data[feature_name]

                y_pred = [np.nan]*len(y_val)
                y_pred = pd.DataFrame(y_pred)
                value = self.mean_mode.predict(feature_name, y_val)
                y_pred = y_pred.fillna(value)
                loss = self.mean_mode.get_mse(y_val, y_pred, feature_name)
                preds.append(value)
                losses.append(loss)

            predicted_mean = np.mean(preds) if type(preds[0]) is float else str(Counte
            y_test = y_test.fillna(predicted_mean)

        else:
            for i in range(len(split_splits)):
                k_copy = split_splits.copy()
                del k_copy[i]
                val = split_splits[i]
                train_i = list(itertools.chain.from_iterable(k_copy))
                val_data = train_data.iloc[val,:]
                new_train_data = train_data.iloc[train_i,:]
```

```
                X_train = new_train_data.drop(feature_name, axis=1)

                self.df_min_max = self.min_max_scalar(X_train)
                X_train = self.scale_transform(X_train)
                y_train = new_train_data[feature_name]

                X_val = val_data.drop(feature_name, axis=1)

                X_val = self.scale_transform(X_val)
                y_val = val_data[feature_name]

                X_train = np.asarray(X_train)
                y_train = np.asarray(y_train)
                X_val = np.asarray(X_val)
                y_val = np.asarray(y_val)

                X_test = self.scale_transform(X_test)
                if model == "linear":
                    self.lin_reg.fit(X_train, y_train, iterations=50)
                    y_pred = self.lin_reg.predict(X_val)
                    y_test = self.lin_reg.predict(X_test)
                    loss = self.lin_reg.get_mse(y_val, y_pred)

                elif model == "ridge":
                    self.ridge.fit(X_train, y_train)
                    y_pred = self.ridge.predict(X_val)
                    y_test = self.ridge.predict(X_test)
                    loss = self.ridge.get_mse(y_val, y_pred)

                elif model == "lasso":
                    self.lasso.fit(X_train, y_train)
                    y_pred = self.lasso.predict(X_val)
                    y_test = self.lasso.predict(X_test)
                    loss = self.lasso.get_mse(y_val, y_pred)

                elif model == "knn":
                    self.knn.fit(X_train, y_train)
                    y_pred = self.knn.predict(X_val)
                    y_test = self.knn.predict(X_test)
                    loss = self.knn.evaluate(y_pred, y_val)

                losses.append(loss)
                preds.append(y_test)

        return pd.DataFrame(y_test_mean, columns=[feature_name]), scaled_loss

    def train_test_data(self, feature_name):
        train_data = self.data[self.full_value_cols + [feature_name]]
        test_data = train_data[train_data[feature_name].isnull()]
        train_data = self.create_full_data(train_data)
        return train_data, test_data
```

```
    def impute_to_main_data(self, new_data, feature_name, max_train_point, count_test_
        indexes = self.data[feature_name].index[self.data[feature_name].apply(np.isnar
        indexes_to_add = [l for l in range(max_train_point, max_train_point+count_test
        for index, index_add in zip(indexes, indexes_to_add):
            self.data[feature_name].iloc[index] = new_data[feature_name].iloc[index_ac

    def pearson_correlation(self, data, main_feature):
        mint = 1e-5
        columns_to_keep = []
        for col in data.columns:
            if col != main_feature:
                r = col, scipy.stats.pearsonr(data[main_feature], data[col])[0]
                if r[1] > 0:
                    columns_to_keep.append(r[0])
        columns_to_keep.append(main_feature)
        data = data[columns_to_keep]
        return data

    def workflow(self):
        missing_value_data, full_value_data = self.missing_value_perc()
        self.full_value_cols = list(full_value_data['feature'])
        subsets = [10, 30, 50, 70, 100]
        lags = [0, 10, 30, 50, 70]
        total_loss = []
        features = []
        self.empty_min_error = []
        self.dict_subset = {}
        for subset,lag in zip(subsets, lags):
            if subset not in self.dict_subset:
                self.dict_subset[subset] = {}
            missing_columns = list(missing_value_data[(missing_value_data["perc"] <= s

            for feature_name in missing_columns:
                try:
                    if feature_name not in self.dict_subset[subset]:
                        self.dict_subset[subset][feature_name] = {}

                    features.append(feature_name)

                    train_data, test_data = self.train_test_data(feature_name)

                    print(train_data.shape, test_data.shape)
                    X_train = train_data.drop(feature_name, axis=1)
                    y_train = train_data[feature_name]
                    multi_class = train_data[feature_name].unique()
                    X_test = test_data.drop(feature_name, axis=1)
                    X_test_copy = X_test.copy()
                    y_test = test_data[feature_name]

                    main_loss = np.iinfo(np.int32(10)).max if eval_type == "Loss" else
                    best_v_test = None
```

```
                    best_model = ''
                    if problem_type not in self.dict_subset[subset][feature_name]:
                        self.dict_subset[subset][feature_name][problem_type] = {}

                    for model in models_to_use:
                        if model not in self.dict_subset[subset][feature_name][problem
                            self.dict_subset[subset][feature_name][problem_type][model

                        y_test, loss = self.cross_validation(train_data, X_test_copy,
                        print("Model: {}, {}: {}".format(model, eval_type, loss))
                        self.dict_subset[subset][feature_name][problem_type][model] =

                    if eval_type == "Accuracy":
                        for key, val in self.dict_subset[subset][feature_name][problem
                            if val >= main_loss:
                                best_y_test = y_test
                                main_loss = val
                                best_model = key

                    if eval_type == "Loss":
                        for key, val in self.dict_subset[subset][feature_name][problem
                            if val <= main_loss:
                                best_y_test = y_test
                                main_loss = val
                                best_model = key

                    X_test = X_test.reset_index()
                    test_data = pd.concat([X_test, best_y_test], axis=1)
                    max_index = len(train_data[feature_name])
                    test_points = len(test_data[feature_name])
                    train_data = train_data.append(test_data, ignore_index=True)
                    self.impute_to_main_data(train_data, feature_name, max_index, test
                    self.full_value_cols.append(feature_name)
                    total_loss.append(main_loss)
                except Exception as e:
                    self.empty_min_error.append(feature_name)



lin_reg = LinearRegression()
ridge = RidgeRegression()
lasso = LassoRegression(max_iters=10)
knn = KNeighbours()
models = [lin_reg, ridge, lasso, knn]


pipeline = Pipeline(data, models, final_answer)
```