University of Essex

CE-888-7-SP DATA SCIENCE AND DECISION MAKING

CAUSAL INFERENCE

ASSIGNMENT-21a3

SUBMITTED BY,

KARAN BHATT

(2112102)

https://github.com/krnbhtt/CE-888

SUBMITTED TO,

DR. ANA MATRAN-FERNANDEZ

TABLE OF CONTENTS

ABSTRACT

For decades, the term causal inference has been a critical research topic in a variety of domains such as computer science, statistics, economics, and public policy. Furthermore, analysing the causal influence from observational data has recently become an interesting topic of study due to the abundance of available data as well as the low budget needs, when compared to randomised controlled trials. In addition, as a result of the rapidly evolving field of machine learning, various estimation methodologies of causal effects for observational data have formed. Furthermore, this research provides an in-depth examination of the "causal inference" technique below the possible output structure. One of them is the framework of causal inference. Furthermore, the processes are divided into two fundamental types based on whether if they need all three assumptions of the corresponding output framework or not. Furthermore, for each categorization, both the current process of enhanced machine learning and the techniques of classical statistics are discussed and compared. The linear regression model is picked in the regression model and its feature relevance is described. Initially, the commonly used benchmark sets of data and open-source scripts are summarised, which generally open the way for researchers to evaluate and use causal inference methodologies. This research will explain the whole training process, from uploading data to preopening it to train by using various models that predict the outcomes of various scenarios.

INTRODUCTION

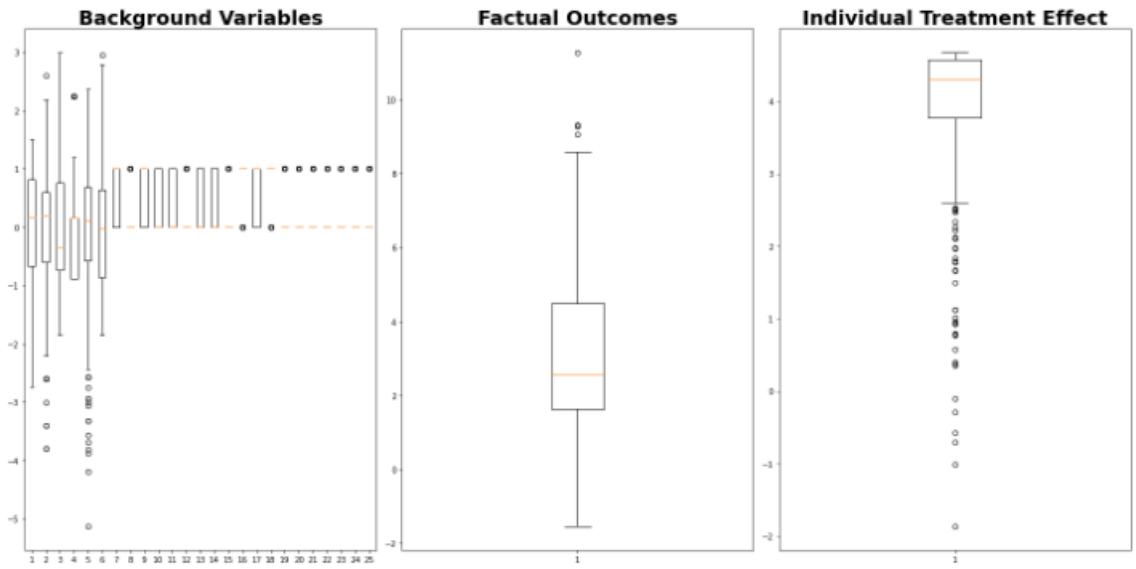
Currently, causality, Language and correlation are largely interchangeable, besides they have wide range of explanation. Furthermore, the term correlation denotes a shared link: if the variables show a rising or declining trend, they are connected. Furthermore, the term causality is used to refer to both the cause and the impact, where the cause is somehow essential for the impact, and the impact is somehow intertwined with the cause. Furthermore, causal inference is the way of formulating a conclusion about the "causal relationship" based on the conditions of occurrence of a result. The difference between the correlation inference and the causal inference is that the computations of the reactions of the outcome variable, even when the consequence is modified. Furthermore, the investigator collects observational data by essentially observing the process without any form of intervention. The investigators have no authority over individuals or treatments, and they just monitor the suggested and gathered data based on their observations. However, using observational data, it is simple to deduce the actions, information, and results of what has occurred. Moreover, the concept causal inference refers to the calculation of a causal effect. Also, the problem is whether just monitored outputs can be used to derive the implications of the proposed system's intervention. This is also known as the fundamental issue of causal inference. Also, the primary goal of this report is to identify the fundamental causal inference data sets, as well as the dataset's features and essential performance metrics. Moreover, the whole report is built on developing a prediction machine learning prediction system to predict whether child is healthy / under treatment, family got support or not (IHDP), whether person have got job training or not. This report is remarkable because, by using this, it would be easy to predict whether a family got a treatment/support or not/they are under control (as per IHDP dataset), or the person is currently employed / did he get a training from NSWP or not, and is he employed or not (as per job data). This report will also discuss how to become acquainted with, in addition to well-established types of causal inference estimation methods, as well as a thorough explanation of the important properties. More than that, this report will help to acquire an instinct for how causal inference relates to machine learning problems. Furthermore, in this assignment, I will load, clean, and investigate the data sets which will be discussed in the data section. In this assignment I have used Liner Regressor and Random Forest Regressor models and explained feature importance, why these models are chosen.

Data:

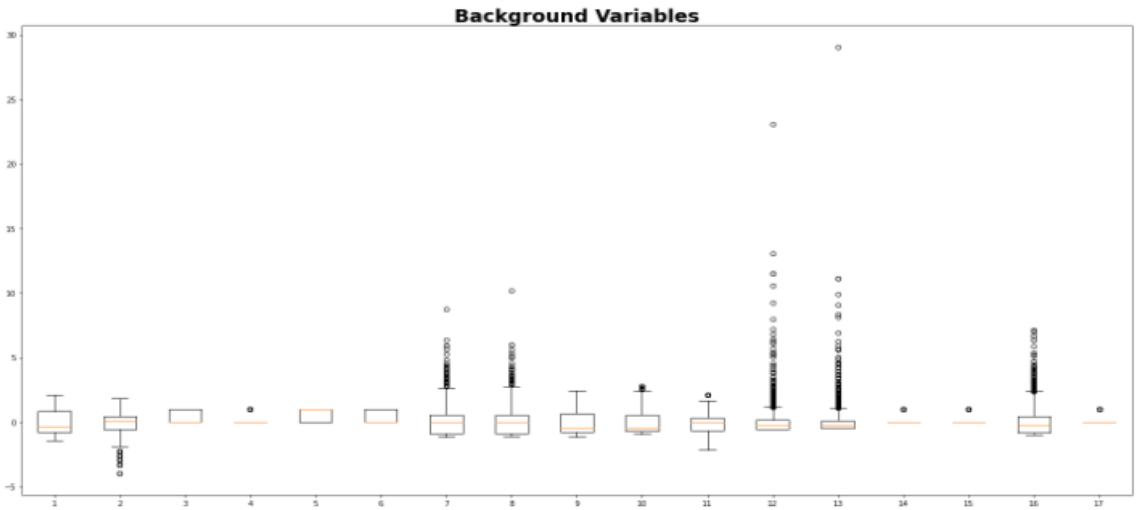
The linear regression model is mostly used to predict the value of a variable based on other factors. While pre-processing the data I found that There are no null values or zero values in both IHDP and job datasets.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3212 entries, 0 to 3211
Data columns (total 17 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       3212 non-null   float64
 1   1       3212 non-null   float64
 2   2       3212 non-null   float64
 3   3       3212 non-null   float64
 4   4       3212 non-null   float64
 5   5       3212 non-null   float64
 6   6       3212 non-null   float64
 7   7       3212 non-null   float64
 8   8       3212 non-null   float64
 9   9       3212 non-null   float64
 10  10      3212 non-null   float64
 11  11      3212 non-null   float64
 12  12      3212 non-null   float64
 13  13      3212 non-null   float64
 14  14      3212 non-null   float64
 15  15      3212 non-null   float64
 16  16      3212 non-null   float64
dtypes: float64(17)
memory usage: 426.7 KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3212 entries, 0 to 3211
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       3212 non-null   uint8
dtypes: uint8(1)
memory usage: 3.3 KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3212 entries, 0 to 3211
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       3212 non-null   uint8
dtypes: uint8(1)
memory usage: 3.3 KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3212 entries, 0 to 3211
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       3212 non-null   uint8
```

therefore, no pre-processing is needed in regards to encoding and filling Nan rows.

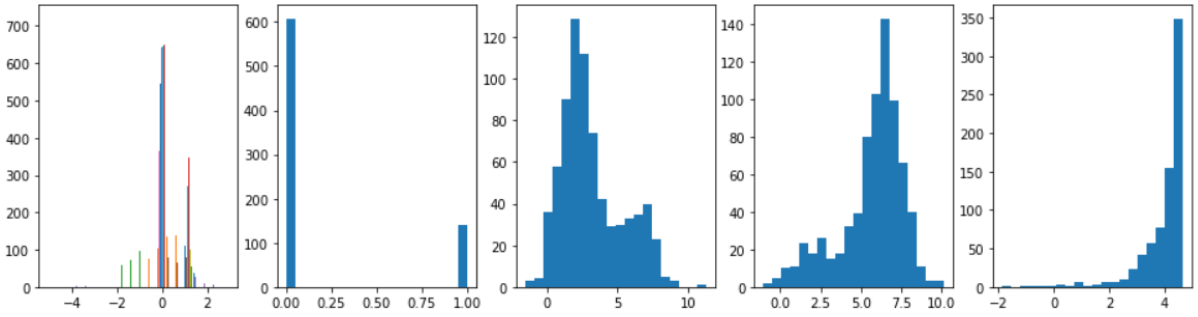


Boxplot of IHDP dataset



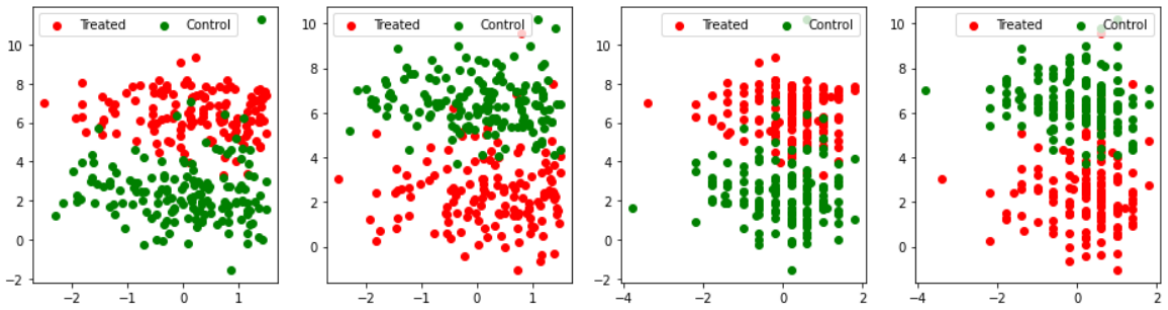
Boxplot of Job Dataset

But There does appear to be quite a few outliers in the IHDP and job dataset (as seen by the values lying outside the minimum and maximum of each box and whisker plot), especially in the non-binary variables. But both datasets vast number of outliers in the background variables, thus a similar approach of experimenting with standardization will need to be taken. We could also investigate random forest regression models that should internally deal with any outliers and reduce the chance of our results being skewed or bias in any way.



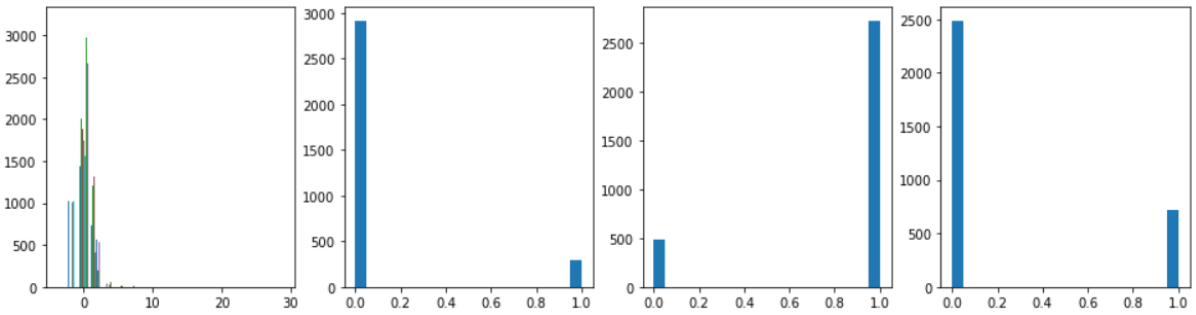
IHDP histogram

The background variables X, although unbalanced, seem to demonstrate a typical shape given the dataset size, the same goes for both the factual and counter factual outcomes. The treatment variable is clearly represented as a binary, 0 being no treatment and 1 being treatment given. The individual treatment effect seems to be extremely negatively skewed although most of the values are around 4 which seems correct given our average treatment effect of 4.02



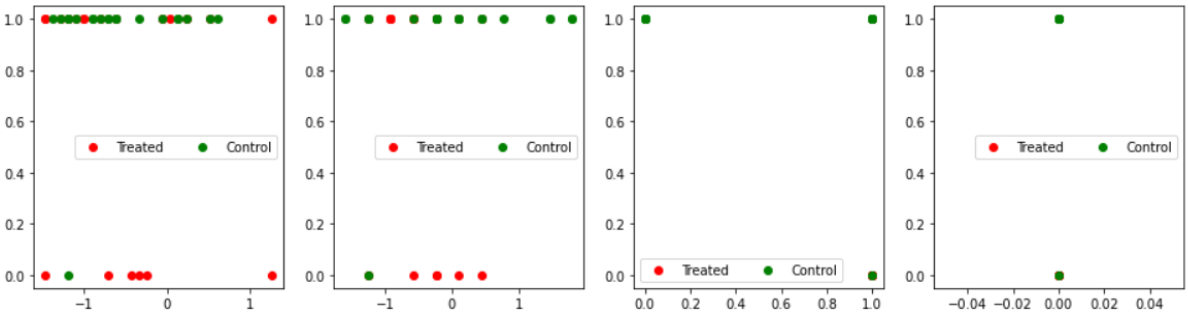
IHDP Scatterplot

Looking at the above graphs of the first two background variables (factual and counterfactual) measured against their outcomes with both the treated and control groups, we can observe some clear effects of the treatment which hopefully means they can be measured as we train and test our regression models.



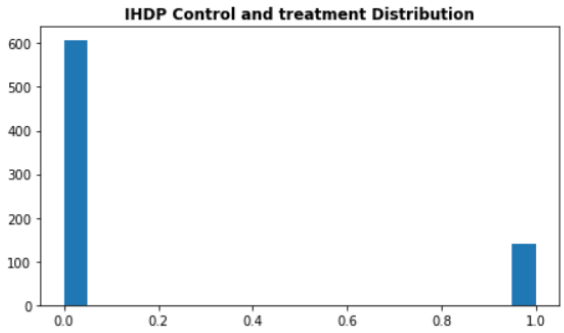
Jobs Histogram

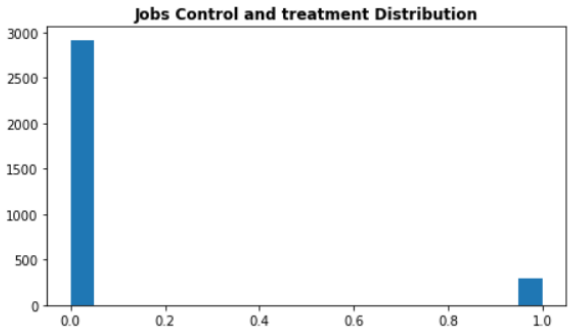
Similarly, to IHDP, the background variables in Jobs seem to be relatively unbalanced, however again demonstrate the typical shape for the relative dataset size. All of t, y, and e represent binary values of either 0 or 1 with all graphs being unbalanced in their distributions. What I mean by this is there is a lot more of one result then there is the other (More 1's than 0's or vice versa)



Jobs Scatterplot

As opposed to IHDP jobs outcomes is recorded as a binary variable thus only plotting scatter points on 0 and 1. It is significantly harder to spot obvious effects given the 4 graphs above, however this could just be a coincidence given the background variables we have chosen to plot.





The above graphs confirm our need to use X-learner. There is clear imbalance towards the treatment and control groups in both datasets, hopefully X-learner should be able to deal with this when calculating our CATE value.

Splitting the Data

I'm looking here to split the datasets with the same ratios of 80/20, where 80% of data will be used as train data and 20% data would be the test data and then create a function that given the parameters that are used to call it, return standardized data.

IHDP_x_train, IHDP_x_test, IHDP_t_train, IHDP_t_test, IHDP_yf_train, IHDP_yf_test, IHDP_ite_train, IHDP_ite_test = train_test_split(IHDP_x, IHDP_t, IHDP_yf, IHDP_ite, test_size=0.2)

Jobs_x_train, Jobs_x_test, Jobs_t_train, Jobs_t_test, Jobs_y_train, Jobs_y_test, Jobs_e_train, Jobs_e_test = train_test_split(Jobs_x, Jobs_t, Jobs_y, Jobs_e, test_size=0.2)

Now we can start data modelling and training.

Methodology:

To solve this causal inference problem, I have used linear regression and random forest regressor. The linear regression model is mostly used to predict the value of a variable based on other variables. Moreover, utilising the methods of cross-validation, dimensionality reduction, and regularisation, linear regression may effectively handle overfitting. And the random forest regressor helps to internally deal with any outliers and reduce the chance of our results being skewed or bias in any way. However, for this case Linear regression and random forest regression both are the regressor tools.

In [26]:
```
# Simple Random forest trained using the IHDP features + treatment
random_forest_IHDP = RandomForestRegressor()
Concatenated_XT_train_IHDP = np.concatenate([IHDP_x_train, IHDP_t_train], axis=1)
random_forest_IHDP.fit(Concatenated_XT_train_IHDP, IHDP_yf_train.flatten())

# Simple Random forest trained using the Jobs features + treatment
random_forest_Jobs = RandomForestRegressor()
Concatenated_XT_train_Jobs = np.concatenate([Jobs_x_train, Jobs_t_train], axis=1)
random_forest_Jobs.fit(Concatenated_XT_train_Jobs, Jobs_y_train.flatten())
```

Out[26]:
```
RandomForestRegressor()
```

Now we predict Y0 and Y1 given t=0 and t=1 respectively for both datasets

In [27]:
```
# t=0 - IHDP
Concatenated_XT_zeros_IHDP = np.concatenate([IHDP_x_test, np.zeros_like(IHDP_t_test)], axis=1)
Y0_IHDP = random_forest_IHDP.predict(Concatenated_XT_zeros_IHDP)
# t=1 - IHDP
Concatenated_XT_ones_IHDP = np.concatenate([IHDP_x_test, np.ones_like(IHDP_t_test)], axis=1)
Y1_IHDP = random_forest_IHDP.predict(Concatenated_XT_ones_IHDP)


# t=0 - Jobs
Concatenated_XT_zeros_Jobs = np.concatenate([Jobs_x_test, np.zeros_like(Jobs_t_test)], axis=1)
Y0_Jobs = random_forest_Jobs.predict(Concatenated_XT_zeros_Jobs)
# t=1 - Jobs
Concatenated_XT_ones_Jobs = np.concatenate([Jobs_x_test, np.ones_like(Jobs_t_test)], axis=1)
Y1_Jobs = random_forest_Jobs.predict(Concatenated_XT_ones_Jobs)
```

Now we can calculate our effect estimates as Y1-Y0

In [28]:
```
#ITEs for IHDP dataset
Effect_Estimates_IHDP = Y1_IHDP - Y0_IHDP
#ITEs for Jobs dataset
Effect_Estimates_Jobs = Y1_Jobs - Y0_Jobs
```

Simple Random Forest regression Model Building and Training

In [29]:

```python
# Functions Taken from Lab 4
def pehe(effect_true, effect_pred):
    """
    Precision in Estimating the Heterogeneous Treatment Effect (PEHE)
    :param effect_true: true treatment effect value
    :param effect_pred: predicted treatment effect value
    :return: PEHE
    """
    return np.abs(np.mean(effect_pred) - np.mean(effect_true))

def abs_ate(effect_true, effect_pred):
    """
    Absolute error for the Average Treatment Effect (ATE)
    :param effect_true: true treatment effect value
    :param effect_pred: predicted treatment effect value
    :return: absolute error on ATE
    """
    return np.sqrt(np.mean((effect_true - effect_pred)**2))
```

In [30]:

```python
# Function taken from D. Machlanski (Unit Supervisor) - https://github.com/dmachlanski
def abs_att(effect_pred, yf, t, e):
    """
    Absolute error for the Average Treatment Effect on the Treated
    :param effect_pred: predicted treatment effect value
    :param yf: factual (observed) outcome
    :param t: treatment status (treated/control)
    :param e: whether belongs to the experimental group
    :return: absolute error on ATT
    """
    att_true = np.mean(yf[t > 0]) - np.mean(yf[(1 - t + e) > 1])
    att_pred = np.mean(effect_pred[(t + e) > 1])

    return np.abs(att_pred - att_true)

def policy_risk(effect_pred, yf, t, e):
    """
    Computes the risk of the policy defined by predicted effect
    :param effect_pred: predicted treatment effect value
    :param yf: factual (observed) outcome
    :param t: treatment status (treated/control)
    :param e: whether belongs to the experimental group
    :return: policy risk
    """
    # Consider only the cases for which we have experimental data (i.e., e > 0)
    t_e = t[e > 0]
    yf_e = yf[e > 0]
    effect_pred_e = effect_pred[e > 0]

    if np.any(np.isnan(effect_pred_e)):
        return np.nan

    policy = effect_pred_e > 0.0
    treat_overlap = (policy == t_e) * (t_e > 0)
    control_overlap = (policy == t_e) * (t_e < 1)

    if np.sum(treat_overlap) == 0:
        treat_value = 0
    else:
        treat_value = np.mean(yf_e[treat_overlap])

    if np.sum(control_overlap) == 0:
        control_value = 0
    else:
        control_value = np.mean(yf_e[control_overlap])

    pit = np.mean(policy)
    policy_value = pit * treat_value + (1.0 - pit) * control_value

    return 1.0 - policy_value
```

Performance evaluation of random forest regressor model

In [31]:

```python
ATE_IHDP = abs_ate(IHDP_ite_test, Effect_Estimates_IHDP)
PEHE_IHDP = pehe(IHDP_ite_test, Effect_Estimates_IHDP)
```

In [32]:

```python
ATT_Jobs = abs_att(Effect_Estimates_Jobs, Jobs_y_test.flatten(), Jobs_t_test.flatten(), Jobs_e_test.flatten())
Rpol = policy_risk(Effect_Estimates_Jobs, Jobs_y_test.flatten(), Jobs_t_test.flatten(), Jobs_e_test.flatten())
```

In [33]:

```python
results = []
results.append(['RF', ATE_IHDP, PEHE_IHDP])
cols = ['Method', 'ATE test', 'PEHE test']
df_First = pd.DataFrame(results, columns=cols)
df_First
```

Out[33]:

	Method	ATE test	PEHE test
0	RF	1.245153	0.190956

In [34]:

```python
results_ = []
results_.append(['RF', ATT_Jobs, Rpol])
cols_ = ['Method', 'ATT test', 'Risk Policy']
df_Second = pd.DataFrame(results_, columns=cols_)
df_Second
```

Out[34]:

	Method	ATT test	Risk Policy
0	RF	0.026039	0.219669

Hyper parameter Tuning:

Here we have used random search parameters, using 10-fold cross validation.

Random Search IHDP ¶

In [35]:
```
Random_Forest_Optimal_IHDP = RandomForestRegressor()

random_grid = {'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}

# Random search of parameters, using 10 fold cross validation,
# search across 100 different combinations, and use all available cores
RandomForest_random_IHDP = RandomizedSearchCV(estimator = Random_Forest_Optimal_IHDP, param_distributions = random_grid,
# Fit the random search model
RandomForest_random_IHDP.fit(Concatenated_XT_train_IHDP, IHDP_yf_train.flatten())
RandomForest_random_IHDP.best_params_
```

```
Fitting 10 folds for each of 100 candidates, totalling 1000 fits
```

Out[35]:
```
{'n_estimators': 1800,
 'min_samples_split': 10,
 'min_samples_leaf': 2,
 'max_features': 'auto',
 'max_depth': 10,
 'bootstrap': True}
```

Random Search Jobs

In [36]:
```
Random_Forest_Optimal_Jobs = RandomForestRegressor()

random_grid = {'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}


# Random search of parameters, using 10 fold cross validation,
# search across 100 different combinations, and use all available cores
RandomForest_random_Jobs = RandomizedSearchCV(estimator = Random_Forest_Optimal_Jobs, param_distributions = random_grid,
# Fit the random search model
RandomForest_random_Jobs.fit(Concatenated_XT_train_Jobs, Jobs_y_train.flatten())
RandomForest_random_Jobs.best_params_
```

```
Fitting 10 folds for each of 100 candidates, totalling 1000 fits
```

Out[36]:
```
{'n_estimators': 800,
 'min_samples_split': 10,
 'min_samples_leaf': 4,
 'max_features': 'sqrt',
 'max_depth': 30,
 'bootstrap': True}
```

Grid Search IHDP

Using the results, we can construct our parameter grid for grid search with values more in the ballpark of what we're looking for

[37]:
```
Benchmark_IHDP = RandomForestRegressor()
# Our parameter Grid uses values centered around the parameters found from the random search
param_grid = {
    'bootstrap': [True],
    'max_depth': [8, 10, 12],
    'max_features': ['auto', 'sqrt'],
    'min_samples_leaf': [2, 3, 4],
    'min_samples_split': [2, 4, 6],
    'n_estimators': [200, 400, 600]
}

GridSearch_RF_IHDP = GridSearchCV(estimator = Benchmark_IHDP, param_grid = param_grid, cv = 10, n_jobs = -1, verbose
GridSearch_RF_IHDP.fit(Concatenated_XT_train_IHDP, IHDP_yf_train.flatten())
GridSearch_RF_IHDP.best_params_
```

```
Fitting 10 folds for each of 162 candidates, totalling 1620 fits
```

Out[37]:
```
{'bootstrap': True,
 'max_depth': 8,
 'max_features': 'auto',
 'min_samples_leaf': 3,
 'min_samples_split': 4,
 'n_estimators': 600}
```

Grid Search Jobs

[38]:
```
Benchmark_Jobs = RandomForestRegressor()
# Our parameter Grid uses values centered around the parameters found from the random search
param_grid = {
    'bootstrap': [True],
    'max_depth': [None, 2, 4],
    'max_features': ['auto', 'sqrt'],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [2, 4, 6],
    'n_estimators': [1800, 2000, 2200]
}

GridSearch_RF_Jobs = GridSearchCV(estimator = Benchmark_Jobs, param_grid = param_grid, cv = 10, n_jobs = -1, verbose
GridSearch_RF_Jobs.fit(Concatenated_XT_train_Jobs, Jobs_y_train.flatten())
GridSearch_RF_Jobs.best_params_
```

```
Fitting 10 folds for each of 162 candidates, totalling 1620 fits
```

Out[38]:
```
{'bootstrap': True,
 'max_depth': None,
 'max_features': 'sqrt',
 'min_samples_leaf': 5,
 'min_samples_split': 2,
 'n_estimators': 1800}
```

Now I am performing feature importance using random forest regression method and plot a graph. Here is the graph of importance features of both IHDP and Job datasets.

In IHDP dataset we can clearly see that column 25 is a most important feature. Column 25 is our concatenated treatment data. In this dataset feature importance is extremely skewed between column no. 25 & 5

In Jobs dataset feature importance are less skewed than IHDP dataset. We can clearly see that between column 12 & 7. Moreover, we can see that our treatment variable is placed on column 12.

Linear Regressor:

In this section we will start with the simple linear regressor model like random forest regressor. Here every binary feature will be standardized.

In [113]:
```
temp_X_IHDP = pd.DataFrame(IHDP_x_train)
temp_X_test_IHDP = pd.DataFrame(IHDP_x_test)
#temp_X_IHDP.head()
temp_yf_IHDP = pd.DataFrame(IHDP_yf_train)
#temp_yf_IHDP.head()
#[temp_X_IHDP[cols].unique() for cols in temp_X_IHDP]

temp_X_Jobs = pd.DataFrame(Jobs_x_train)
temp_X_test_Jobs = pd.DataFrame(Jobs_x_test)
#temp_X_Jobs.head()
#[temp_X_Jobs[cols].unique() for cols in temp_X_Jobs]
temp_X_Jobs.head()
```

Out[113]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	-0.332011	0.783281	0.0	0.0	1.0	0.0	0.239950	0.433634	-0.435556	-0.499853	0.743585	-0.094318	0.035185	0.0	0.0	0.331067	0.0
1	-1.367005	-1.261057	0.0	0.0	0.0	1.0	-1.063009	-1.013806	-1.117857	-0.927646	-1.201098	-0.556341	-0.516644	0.0	0.0	-0.934404	0.0
2	1.173434	0.101835	0.0	0.0	1.0	0.0	1.114277	0.857365	1.127008	1.038605	-0.022502	0.674367	0.380960	0.0	0.0	0.851974	0.0
3	-0.802483	0.101835	1.0	0.0	1.0	0.0	-0.085212	-0.263007	-0.785284	-0.743407	-0.022502	-0.286221	-0.351938	0.0	0.0	-0.119606	0.0
4	-0.237921	0.783281	0.0	0.0	1.0	0.0	0.059304	0.146360	-0.357691	-0.439662	0.743585	-0.207221	-0.151779	0.0	0.0	0.160358	0.0

In IHDP dataset we can see that column 0-5 require standard scaling. From past model we know that our outcome column requires it.

In jobs dataset there are many columns which requires standard scaling rest are binary, so they do not require it.

In [114]:
```
# IHDP
# Scale the first 6 columns of our features (all non binary)
temp_X_IHDP.iloc[:, 0:5] = StandardScaler().fit_transform(temp_X_IHDP.iloc[:, 0:5])
temp_X_test_IHDP.iloc[:, 0:5] = StandardScaler().fit_transform(temp_X_test_IHDP.iloc[:, 0:5])
# Scale our outcomes column
IHDP_yf_train_Standardized = StandardScaler().fit_transform(temp_yf_IHDP)
#temp_X_IHDP.head()
IHDP_x_train_Standardized = temp_X_IHDP.to_numpy()
IHDP_x_test_Standardized = temp_X_test_IHDP.to_numpy()


# Jobs
# Scale columns 0,1,6,7,8,9,10,11,12,15 (all non binary)
temp_X_Jobs.iloc[:, [0,1,6,7,8,9,10,11,12,15]] = StandardScaler().fit_transform(temp_X_Jobs.iloc[:, [0,1,6,7,8,9,10,11,1
temp_X_test_Jobs.iloc[:, [0,1,6,7,8,9,10,11,12,15]] = StandardScaler().fit_transform(temp_X_test_Jobs.iloc[:, [0,1,6,7,8
Jobs_x_train_Standardized = temp_X_Jobs.to_numpy()
Jobs_x_test_Standardized = temp_X_test_Jobs.to_numpy()
temp_X_Jobs.head()
```

Out[114]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	-0.333640	0.793828	0.0	0.0	1.0	0.0	0.243889	0.433519	-0.437071	-0.501097	0.756902	-0.091168	0.033472	0.0	0.0	0.340166	0.0
1	-1.365976	-1.248920	0.0	0.0	0.0	1.0	-1.065070	-1.007946	-1.116929	-0.926842	-1.193638	-0.555928	-0.509009	0.0	0.0	-0.940238	0.0
2	1.167940	0.112912	0.0	0.0	1.0	0.0	1.122241	0.855500	1.119896	1.029997	-0.011492	0.682069	0.373390	0.0	0.0	0.867221	0.0
3	-0.802883	0.112912	1.0	0.0	1.0	0.0	-0.082771	-0.260246	-0.785548	-0.743485	-0.011492	-0.284209	-0.347093	0.0	0.0	-0.115824	0.0
4	-0.239791	0.793828	0.0	0.0	1.0	0.0	0.062411	0.147430	-0.359486	-0.441194	0.756902	-0.204740	-0.150325	0.0	0.0	0.167443	0.0

Now let's start with building the simple linear regressor model.

In [116]:
```
# Concatenate the treatment and feature columns for both datasets
Concatenated_XT_train_IHDP = np.concatenate([IHDP_x_train_Standardized, IHDP_t_train], axis=1)
Linear_Regressor_IHDP = LinearRegression().fit(Concatenated_XT_train_IHDP, IHDP_yf_train_Standardized.flatten())

Concatenated_XT_train_Jobs = np.concatenate([Jobs_x_train_Standardized, Jobs_t_train], axis=1)
Linear_Regressor_Jobs = LinearRegression().fit(Concatenated_XT_train_Jobs, Jobs_y_train.flatten())

```

In [122]:
```
# t=0 - IHDP
Concatenated_XT_zeros_IHDP = np.concatenate([IHDP_x_test_Standardized, np.zeros_like(IHDP_t_test)], axis=1)
Y0_IHDP = Linear_Regressor_IHDP.predict(Concatenated_XT_zeros_IHDP)
# t=1 - IHDP
Concatenated_XT_ones_IHDP = np.concatenate([IHDP_x_test_Standardized, np.ones_like(IHDP_t_test)], axis=1)
Y1_IHDP = Linear_Regressor_IHDP.predict(Concatenated_XT_ones_IHDP)


# t=0 - Jobs
Concatenated_XT_zeros_Jobs = np.concatenate([Jobs_x_test_Standardized, np.zeros_like(Jobs_t_test)], axis=1)
Y0_Jobs = Linear_Regressor_Jobs.predict(Concatenated_XT_zeros_Jobs)
# t=1 - Jobs
Concatenated_XT_ones_Jobs = np.concatenate([Jobs_x_test_Standardized, np.ones_like(Jobs_t_test)], axis=1)
Y1_Jobs = Linear_Regressor_Jobs.predict(Concatenated_XT_ones_Jobs)
```

In [123]:
```
#ITEs for IHDP dataset
Effect_Estimates_IHDP_LR = Y1_IHDP - Y0_IHDP
#ITEs for Jobs dataset
Effect_Estimates_Jobs_LR = Y1_Jobs - Y0_Jobs
```

So now our effect estimates are built so we can continue with metric evaluation.

In [124]:
```
ATE_IHDP_LR = abs_ate(IHDP_ite_test, Effect_Estimates_IHDP_LR)
PEHE_IHDP_LR = pehe(IHDP_ite_test, Effect_Estimates_IHDP_LR)
ATT_Jobs_LR = abs_att(Effect_Estimates_Jobs_LR, Jobs_y_test.flatten(), Jobs_t_test.flatten(), Jobs_e_test.flatten())
Rpol_LR = policy_risk(Effect_Estimates_Jobs_LR, Jobs_y_test.flatten(), Jobs_t_test.flatten(), Jobs_e_test.flatten())
```

In [125]:
```
# Metrics for IHDP dataset
results = []
results.append(['RF', ATE_IHDP, PEHE_IHDP])
results.append(['Optimized RF', ATE_IHDP_Optimized, PEHE_IHDP_Optimized])
results.append(['LR', ATE_IHDP_LR, PEHE_IHDP_LR])
cols = ['Method', 'ATE test', 'PEHE test']
df_First = pd.DataFrame(results, columns=cols)
df_First
```

Out[125]:

	Method	ATE test	PEHE test
0	RF	1.305712	0.116149
1	Optimized RF	1.254617	0.137322
2	LR	2.401287	2.263552

In [126]:
```
# Metrics for Jobs Dataset
results_ = []
results_.append(['RF', ATT_Jobs, Rpol])
results_.append(['Optimized RF', ATT_Jobs_Optimized, Rpol_Optimized])
results_.append(['LR', ATT_Jobs_LR, Rpol_LR])
cols_ = ['Method', 'ATT test', 'Risk Policy']
df_Second = pd.DataFrame(results_, columns=cols_)
df_Second
```

Out[126]:

	Method	ATT test	Risk Policy
0	RF	0.063641	0.202296
1	Optimized RF	0.033940	0.217834
2	LR	0.006004	0.224490

For hyperparameter tuning, we can execute a grid search using the selection of parameters available to us to find the best performing linear regression model

Grid Search IHDP

In [128]:
```
LinearRegressor_IHDP = LinearRegression()
param_grid = {'fit_intercept': [True, False],
 'positive': [True, False]}
LinearRegressor_Grid_IHDP = GridSearchCV(estimator = LinearRegressor_IHDP, param_grid=param_grid, cv = 10, verbose=1
LinearRegressor_Grid_IHDP.fit(Concatenated_XT_train_IHDP, IHDP_yf_train_Standardized.flatten())
LinearRegressor_Grid_IHDP.best_params_
```

```
Fitting 10 folds for each of 4 candidates, totalling 40 fits
```

Out[128]:
```
{'fit_intercept': True, 'positive': True}
```

Grid Search Jobs

In [129]:
```
LinearRegressor_Jobs = LinearRegression()
param_grid = {'fit_intercept': [True, False],
 'positive': [True, False]}
LinearRegressor_Grid_Jobs = GridSearchCV(estimator = LinearRegressor_Jobs, param_grid=param_grid, cv = 10, verbose=1
LinearRegressor_Grid_Jobs.fit(Concatenated_XT_train_Jobs, Jobs_y_train.flatten())
LinearRegressor_Grid_Jobs.best_params_
```

So, our grid search has been executed so let's move towards performance evaluation side combined with 10-fold cross validation.

In [130]:
```
LR_IHDP_Optimized = LinearRegression(fit_intercept=True, positive=True)
LR_IHDP_Optimized.fit(Concatenated_XT_train_IHDP, IHDP_yf_train_Standardized.flatten())

# t=0 - IHDP
Concatenated_XT_zeros_IHDP = np.concatenate([IHDP_x_test_Standardized, np.zeros_like(IHDP_t_test)], axis=1)
Y0_IHDP = LR_IHDP_Optimized.predict(Concatenated_XT_zeros_IHDP)
# t=1 - IHDP
Concatenated_XT_ones_IHDP = np.concatenate([IHDP_x_test_Standardized, np.ones_like(IHDP_t_test)], axis=1)
Y1_IHDP = LR_IHDP_Optimized.predict(Concatenated_XT_ones_IHDP)

LR_Jobs_Optimized = LinearRegression(fit_intercept=True, positive=False)
LR_Jobs_Optimized.fit(Concatenated_XT_train_Jobs, Jobs_y_train.flatten())

# t=0 - Jobs
Concatenated_XT_zeros_Jobs = np.concatenate([Jobs_x_test_Standardized, np.zeros_like(Jobs_t_test)], axis=1)
Y0_Jobs = LR_Jobs_Optimized.predict(Concatenated_XT_zeros_Jobs)
# t=1 - Jobs
Concatenated_XT_ones_Jobs = np.concatenate([Jobs_x_test_Standardized, np.ones_like(Jobs_t_test)], axis=1)
Y1_Jobs = LR_Jobs_Optimized.predict(Concatenated_XT_ones_Jobs)

#ITEs for IHDP dataset
Effect_Estimates_IHDP_LR_Optimized = Y1_IHDP - Y0_IHDP
#ITEs for Jobs dataset
Effect_Estimates_Jobs_LR_Optimized = Y1_Jobs - Y0_Jobs
```

In [131]:
```
ATE_IHDP_LR_Optimized = abs_ate(IHDP_ite_test, Effect_Estimates_IHDP_LR_Optimized)
PEHE_IHDP_LR_Optimized = pehe(IHDP_ite_test, Effect_Estimates_IHDP_LR_Optimized)
ATT_Jobs_LR_Optimized = abs_att(Effect_Estimates_Jobs_LR_Optimized, Jobs_y_test.flatten(), Jobs_t_test.flatten(), Jobs_e
Rpol_LR_Optimized = policy_risk(Effect_Estimates_Jobs_LR_Optimized, Jobs_y_test.flatten(), Jobs_t_test.flatten(), Jobs_e
```

In [132]:
```
# Metrics for IHDP dataset
results = []
results.append(['RF', ATE_IHDP, PEHE_IHDP])
results.append(['Optimized RF', ATE_IHDP_Optimized, PEHE_IHDP_Optimized])
results.append(['LR', ATE_IHDP_LR, PEHE_IHDP_LR])
results.append(['Optimized LR', ATE_IHDP_LR_Optimized, PEHE_IHDP_LR_Optimized])
cols = ['Method', 'ATE test', 'PEHE test']
df_First = pd.DataFrame(results, columns=cols)
df_First
```

Out[132]:

	Method	ATE test	PEHE test
0	RF	1.305712	0.116149
1	Optimized RF	1.254617	0.137322
2	LR	2.401287	2.263552
3	Optimized LR	2.410418	2.273235

In [134]:
```
# Metrics for Jobs Dataset
results_ = []
results_.append(['RF', ATT_Jobs, Rpol])
results_.append(['Optimized RF', ATT_Jobs_Optimized, Rpol_Optimized])
results_.append(['LR', ATT_Jobs_LR, Rpol_LR])
results_.append(['Optimized LR', ATT_Jobs_LR_Optimized, Rpol_LR_Optimized])
cols_ = ['Method', 'ATT test', 'Risk Policy']
df_Second = pd.DataFrame(results_, columns=cols_)
df_Second
```

Out[134]:

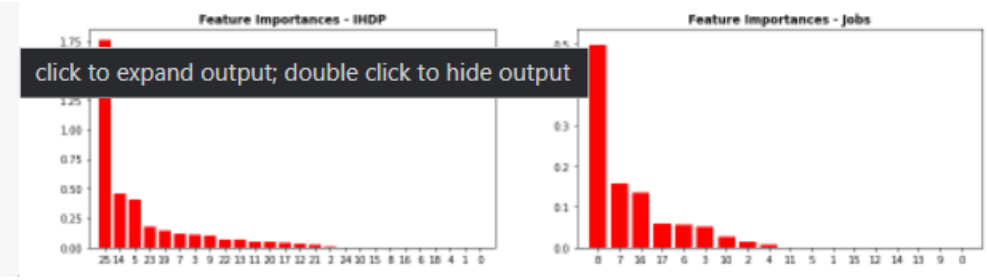
	Method	ATT test	Risk Policy
0	RF	0.063641	0.202295
1	Optimized RF	0.033940	0.217834
2	LR	0.006004	0.224490
3	Optimized LR	0.006004	0.224490

Now finally we can plot and visualise the feature importance of the data, from more useful to less useful. we can recognize feature by their column indices.

In [135]:
```
LR_IHDP_Optimized.fit(Concatenated_XT_train_IHDP, IHDP_yf_train_Standardized.flatten())
features_IHDP = list(Concatenated_XT_train_IHDP)
importances_IHDP = LR_IHDP_Optimized.coef_
indices_IHDP = np.argsort(importances_IHDP)[::-1]
print("Feature Importances - IHDP")
for f in range(Concatenated_XT_train_IHDP.shape[1]):
    print("%d. %s (%f)" % (f + 1, "Feature Column: "+ str(indices_IHDP[f]), importances_IHDP[indices_IHDP[f]]))


LR_Jobs_Optimized.fit(Concatenated_XT_train_Jobs, Jobs_y_train.flatten())
features_Jobs = list(Concatenated_XT_train_Jobs)
importances_Jobs = LR_Jobs_Optimized.coef_
indices_Jobs = np.argsort(importances_Jobs)[::-1]
print("")
print("Feature Importances - Jobs")
for f in range(Concatenated_XT_train_Jobs.shape[1]):
    print("%d. %s (%f)" % (f + 1, "Feature Column: "+ str(indices_Jobs[f]), importances_Jobs[indices_Jobs[f]]))
```

```
Feature Importances - IHDP
1. Feature Column: 25 (1.760401)
2. Feature Column: 14 (0.459039)
3. Feature Column: 5 (0.407445)
4. Feature Column: 23 (0.177242)
5. Feature Column: 19 (0.141710)
6. Feature Column: 7 (0.118741)
7. Feature Column: 3 (0.110800)
8. Feature Column: 9 (0.101924)
9. Feature Column: 22 (0.066579)
10. Feature Column: 13 (0.063097)
11. Feature Column: 11 (0.048673)
12. Feature Column: 20 (0.046622)
13. Feature Column: 17 (0.043350)
14. Feature Column: 12 (0.032215)
15. Feature Column: 21 (0.027382)
16. Feature Column: 2 (0.008747)
17. Feature Column: 24 (0.000000)
18. Feature Column: 10 (0.000000)
19. Feature Column: 15 (0.000000)
20. Feature Column: 8 (0.000000)
21. Feature Column: 16 (0.000000)
22. Feature Column: 6 (0.000000)
23. Feature Column: 18 (0.000000)
24. Feature Column: 4 (0.000000)
25. Feature Column: 1 (0.000000)
26. Feature Column: 0 (0.000000)

Feature Importances - Jobs
1. Feature Column: 8 (0.496983)
2. Feature Column: 7 (0.156823)
3. Feature Column: 16 (0.134872)
4. Feature Column: 17 (0.058395)
5. Feature Column: 6 (0.055442)
6. Feature Column: 3 (0.051712)
7. Feature Column: 10 (0.026582)
8. Feature Column: 2 (0.014205)
9. Feature Column: 4 (0.007310)
10. Feature Column: 11 (-0.006009)
11. Feature Column: 5 (-0.006770)
12. Feature Column: 1 (-0.024295)
13. Feature Column: 15 (-0.062146)
14. Feature Column: 12 (-0.069901)
15. Feature Column: 14 (-0.151532)
16. Feature Column: 13 (-0.155559)
17. Feature Column: 9 (-0.267712)
18. Feature Column: 0 (-0.284215)
```

In [136]:
```
fig, axs = plt.subplots(1, 2, figsize=(16, 4))
axs[0].bar(range(Concatenated_XT_train_IHDP.shape[1]), importances_IHDP[indices_IHDP], color="r", align="center")
axs[0].set_xticks(range(Concatenated_XT_train_IHDP.shape[1]), indices_IHDP)
axs[0].set_xlim([-1, Concatenated_XT_train_IHDP.shape[1]])
axs[0].set_ylim([0, None])
axs[0].set_title("Feature Importances - IHDP", fontsize=12, fontweight="bold")
axs[1].bar(range(Concatenated_XT_train_Jobs.shape[1]), importances_Jobs[indices_Jobs], color="r", align="center")
axs[1].set_xticks(range(Concatenated_XT_train_Jobs.shape[1]), indices_Jobs)
axs[1].set_xlim([-1, Concatenated_XT_train_Jobs.shape[1]])
axs[1].set_ylim([0, None])
axs[1].set_title("Feature Importances - Jobs", fontsize=12, fontweight="bold")
plt.show()
```



In IHDP dataset our feature importance follow similar pattern with the results being extremely skewed and our most important features being, our treatment column, background feature 14, and background feature 5 same as random forest regressor.

In Jobs dataset The skewness of the feature importance has completely changes after using linear regressions, we now have unbalanced importance, with our previously most importance features being much further down the list, we seemingly have increased importance in regards to the treatment feature however

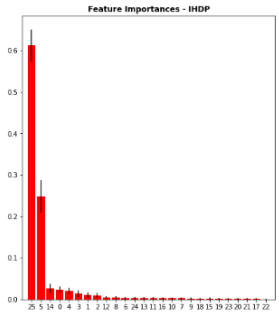
For propensity Score Re-weighting we are going to use a random forest classifier. First we need to train a classifer to predict propensity scores based on background features. We also need a function that calculates sample weights based on these propensity scores. Given equation is as follow: $w_i = \frac{t_i}{e(x_i)} + \frac{1 - t_i}{1 - e(x_i)}$

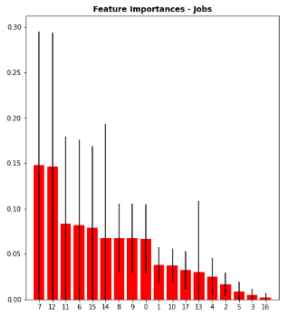
In [138]:
```
def Calc_Weights(ti, ex):
    wi = (ti / ex) + ((1-ti) / (1-ex))
    return wi

Sample_Weights_IHDP = Calc_Weights(ti_IHDP, ex_IHDP)
Sample_Weights_Jobs = Calc_Weights(ti_Jobs, ex_Jobs)
```

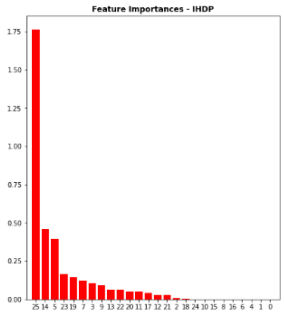
Now we have sample weights so we can train our regressors using the weights.

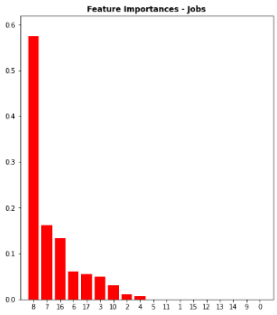
In [139]:
```
# Training variables for random forest
Concatenated_XT_train_IHDP = np.concatenate([IHDP_x_train, IHDP_t_train], axis=1)
Concatenated_XT_train_Jobs = np.concatenate([Jobs_x_train, Jobs_t_train], axis=1)

# Training variables for linear regression
Concatenated_XT_train_IHDP_Standardized = np.concatenate([IHDP_x_train_Standardized, IHDP_t_train], axis=1)
Concatenated_XT_train_Jobs_Standardized = np.concatenate([Jobs_x_train_Standardized, Jobs_t_train], axis=1)

RandomForest_IHDP_IPSW = RandomForestRegressor()
RandomForest_Jobs_IPSW = RandomForestRegressor()
LinearRegressor_IHDP_IPSW = LinearRegression()
LinearRegressor_Jobs_IPSW = LinearRegression()

# Trained regressors
RandomForest_IHDP_IPSW.fit(Concatenated_XT_train_IHDP, IHDP_yf_train.flatten(), sample_weight=Sample_Weights_IHDP)
RandomForest_Jobs_IPSW.fit(Concatenated_XT_train_Jobs, Jobs_y_train.flatten(), sample_weight=Sample_Weights_Jobs)
LinearRegressor_IHDP_IPSW.fit(Concatenated_XT_train_IHDP, IHDP_yf_train.flatten(), sample_weight=Sample_Weights_IHDP)
LinearRegressor_Jobs_IPSW.fit(Concatenated_XT_train_Jobs, Jobs_y_train.flatten(), sample_weight=Sample_Weights_Jobs)
```

Now we will repeat all Random Search, Grid Search, Metric Evaluation, Feature importance (IPSW) and then we will visualise the data again.









Advanced CATE Estimators

In [84]:
```
xl_IHDP = XLearner(models=RandomForestRegressor(), propensity_model=RandomForestClassifier())
xl_Jobs = XLearner(models=RandomForestRegressor(), propensity_model=RandomForestClassifier())

xl_IHDP.fit(IHDP_yf_train, IHDP_t_train.flatten(), X=IHDP_x_train)
xl_Jobs.fit(Jobs_y_train, Jobs_t_train.flatten(), X=Jobs_x_train)

xl_te_test_IHDP = xl_IHDP.effect(IHDP_x_test)
xl_te_test_Jobs = xl_Jobs.effect(Jobs_x_test)
```

In [85]:
```
xl_ate_test = abs_ate(IHDP_ite_test, xl_te_test_IHDP)
xl_pehe_test = pehe(IHDP_ite_test, xl_te_test_IHDP)
xl_att_test = abs_att(xl_te_test_Jobs, Jobs_y_test.flatten(), Jobs_t_test.flatten(), Jobs_e_test.flatten())
xl_rpol = policy_risk(xl_te_test_Jobs.flatten(), Jobs_y_test.flatten(), Jobs_t_test.flatten(), Jobs_e_test.flatten(
```

In [86]:
```
# Metrics for IHDP dataset
results = []
results.append(['RF', ATE_IHDP, PEHE_IHDP])
results.append(['Optimized RF', ATE_IHDP_Optimized, PEHE_IHDP_Optimized])
results.append(['LR', ATE_IHDP_LR, PEHE_IHDP_LR])
results.append(['Optimized LR', ATE_IHDP_LR_Optimized, PEHE_IHDP_LR_Optimized])
results.append(['IPSW RF', ATE_IHDP_RF_IPSW, PEHE_IHDP_RF_IPSW])
results.append(['IPSW LR',ATE_IHDP_LR_IPSW,PEHE_IHDP_LR_IPSW])
results.append(['XL',xl_ate_test ,xl_pehe_test])
cols = ['Method', 'ATE test', 'PEHE test']
df_First = pd.DataFrame(results, columns=cols)
df_First
```

Out[86]:

	Method	ATE test	PEHE test
0	RF	1.245153	0.190956
1	Optimized RF	1.209269	0.196533
2	LR	2.385154	2.274672
3	Optimized LR	2.386931	2.276535
4	IPSW RF	1.192577	0.171409
5	IPSW LR	2.385161	2.274679
6	XL	0.501730	0.143855

In [87]:
```
# Metrics for Jobs Dataset
results_ = []
results_.append(['RF', ATT_Jobs, Rpol])
results_.append(['Optimized RF', ATT_Jobs_Optimized, Rpol_Optimized])
results_.append(['LR', ATT_Jobs_LR, Rpol_LR])
results_.append(['Optimized LR', ATT_Jobs_LR_Optimized, Rpol_LR_Optimized])
results_.append(['IPSW RF',ATT_Jobs_RF_IPSW,Rpol_RF_IPSW])
results_.append(['IPSW LR',ATT_Jobs_LR_IPSW,Rpol_LR_IPSW])
results_.append(['XL',xl_att_test ,xl_rpol])
cols_ = ['Method', 'ATT test', 'Risk Policy']
df_Second = pd.DataFrame(results_, columns=cols_)
df_Second
```

Out[87]:

	Method	ATT test	Risk Policy
0	RF	0.026039	0.219689
1	Optimized RF	0.014877	0.288724
2	LR	0.038637	0.259259
3	Optimized LR	0.038637	0.259259
4	IPSW RF	0.029692	0.206518
5	IPSW LR	0.029424	0.259259
6	XL	0.088744	0.275864

Conclusion:

According to the final metric table of IHDP dataset, XLearner seems to be the most consistent and highly performing model, with the highest ATE and second best PEHE scores as compared to jobs dataset. I would say that, if I had to offer a suggestion to an organization adopting this specific dataset, I would prefer XLearner because of its capability to counterbalanced control to treated data sets, resulting in perhaps the greatest performance overall.

The basic random forest model appears to be the best performing model for this jobs dataset, offering not just the most precise result of treatment impact on the treated, as well as a relatively low risk policy. However, the standard model's ATT is so precise that I feel it is an anomaly, for example if I reran the code, I do not believe I would get the same result. If I had to provide a recommendation to a firm that was using this data set, I would suggest using the linear regressor. It has incredibly less errors on average across three different different versions, demonstrating consistency in performance. Because optimising the hyperparameters is challenging with this model, there is room for the model to improve.

References:

1. Prosperi, M., Guo, Y., Sperrin, M., Koopman, J.S., Min, J.S., He, X., Rich, S., Wang, M., Buchan, I.E. and Bian, J., 2020. Causal inference and counterfactual prediction in machine learning for actionable healthcare. *Nature Machine Intelligence*, *2*(7), pp.369-375.
2. Chen, H., Harinen, T., Lee, J.Y., Yung, M. and Zhao, Z., 2020. Causalml: Python package for causal machine learning. *arXiv preprint arXiv:2002.11631*.
3. Pearl, J., 2019. The seven tools of causal inference, with reflections on machine learning. *Communications of the ACM*, *62*(3), pp.54-60.
4. Tolles, Juliana; Meurer, William J (2016). "Logistic Regression Relating Patient Characteristics to Outcomes". *JAMA*. **316** (5): 533–4.
5. Min, J.S., He, X., Rich, S., Wang, M., Buchan, I.E., and Bian, J., 2020. Causal inference and counterfactual prediction in machine learning for actionable healthcare. *Nature Machine Intelligence*, *2*(7), pp.369-375.
6. Chen, H., Harinen, T., Causalml: Python package for causal machine learning. *arXiv preprint arXiv:2002.11631*.
7. https://www.microsoft.com/en-us/research/group/causal-inference/
8. https://en.wikipedia.org/wiki/Causal_inference
9. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2836213/
10. https://scholar.google.co.uk/scholar?q=causal+inference+scholarly+article&hl=en&as_sdt=0&as_vis=1&oi=scholart
11. **https://www.educba.com/linear-search-in-data-structure/**
12. **https://www.academia.edu/20338819/simple_linear_regressor**