Report for Econometric Competition on Predicting 2019 Individual Income

Team: ╲_😉_╱
Member: Hengyu Kuang

January 4, 2022

Abstract

We build an ensemble model from LightGBM and Transformer based neural net to predict 2019 individual annual pre-tax wage and salary. The error analysis suggests college and graduate students have the highest error, and ages in both tails have unbalanced errors. We find mean square error weighted by education gives lower overall error. We follow the competition rules and add magics in the final prediction. Codes are available in the github page.

1 Preliminary Data Analysis

Data is a subset of 2019 American Community Survey obtained from IPUMS (Ruggles et al. 2021). The goal is to predict individual pre-tax wage and salary using 12 variables. The evaluation metric is mean square error of log income.

The dependent variable, incwage, is non-negative integer and ranges within [0, 999999]. 999999 is encoded as N/A, and 999998 means missing. Neither values are in the samples. Dependent variables are sex, age, martial status (marst), race, birthplace (bpl), language, educational attainment (educ), occupation (occ), weeks worked last year (wkswork1), hours worked per week (uhrswork), and veteran status (vetstat). All independent variables are pre-encoded as integers.

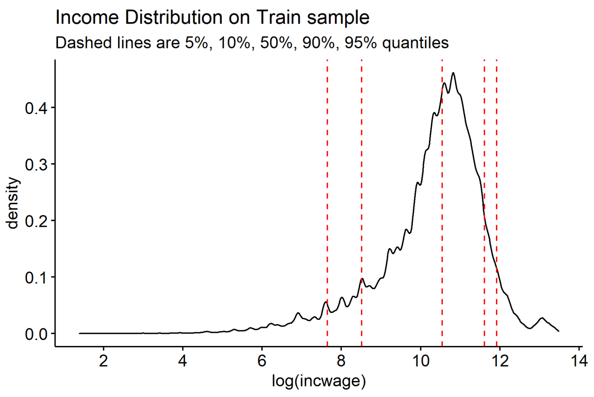
The training data has 318,074 rows, and the test data has 1,272,344 rows. The training data has 1,404 (0.44%) duplicated rows. 1,274 (90.74%) of them have 2 duplicates, and 119 (8.47%) have 3 duplicates. The test data has 86,804 (6.82%) duplicated rows. We expect the independent variables fail to provide enough information to identify the income variable.

Numeric variables are age, hours worked last year (whrswork) , weeks worked last year (wkswork1). They can also be treated as categorical variables as value ranges are fixed integers.

Categorical independent variables are the remaining 9 variables with various levels. Specifically, Birthplace (bpl) has 123 levels, occupation (occ) has 529, and language spoken has 63.

The distribution on income is right-skewed with mean 54,278, median 38,000, and std 68086.31. The top 10% quantile is 110,000, and bottom 10% is 5,000. Figure 1 shows the distribution on log of income. It's approximately normal with mean 10.29 and std 1.299.

Figure 1:



All independent variables in the train and test sets have approximately the same distributions. Both non-parametric Kruskal-Wallis tests and ANOVA fail to reject the null hypothesis that the independent variables in the two datasets have different means.

In the training sample, average high income groups are EU member counties (bpl in 400 range) and minority language speakers. The White (race=1) people has the most observations, but their average income ranks the 4th. Asian populations are in the highest average income group. STEAM majors also have the highest average income.

We highlight income against education and age. Additional plots on comparing variable distributions are in the appendix.

Figure 2 shows people with higher educations have much higher income. More than 5 years of college (educ = 11) account 14.29% of the training sample, and their average income is more than 100,000. College students (educ=10,8,7) ranked second, and are followed by high school graduates (educ=6).

Figure 2:

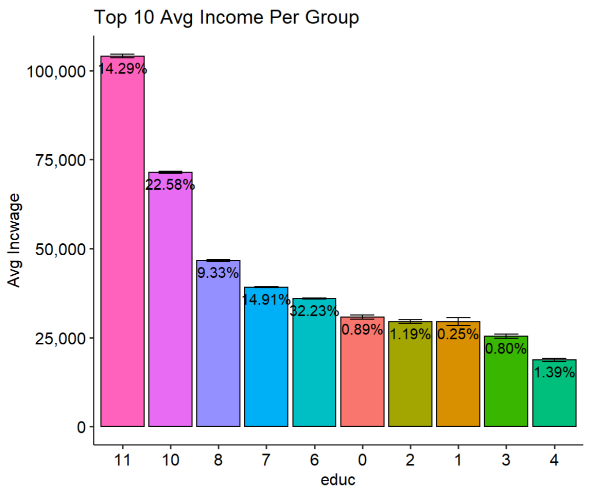
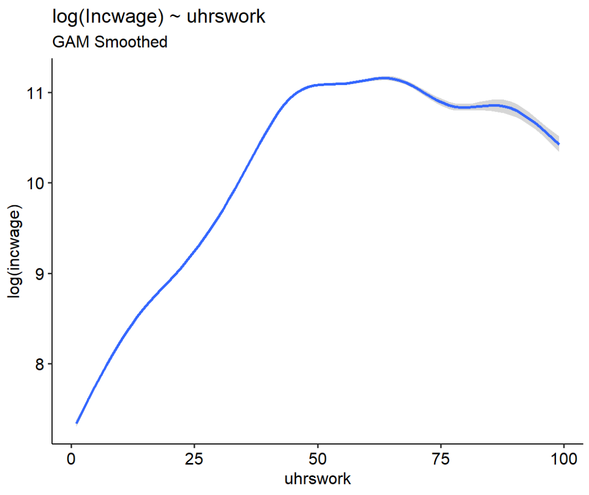


Figure 3 shows young people with age less than 25 tend to have much lower income compared to middle age. However, the senior group has larger variance. We expect young and highly educated people have less income in their early careers.

Figure 3:



2 Models

The categorical variables have large amount of levels, and there are duplicate rows. Instead of using linear models, we use gradient boosting trees and Transformer based neural networks to build the final ensemble model.

2.1 Boosting Trees

We compare Light Gradient Boosting Machine (LightGBM) (Ke et al. 2017), Categorical Boosting (CatBoost) (Dorogush, Ershov, and Gulin 2018), and Extreme Gradient Boosting (XGBoost) (Chen and Guestrin 2016). LightGBM consistently outperforms other tree models by 4% to 15% in subsets of the training data, and in mixed ensemble trees its predictions have much higher weights, so we choose LightGBM.

We use the Python package “AutoGluon” (Erickson et al. 2020) implantation and use Bayesian optimization to search hyper-parameters.

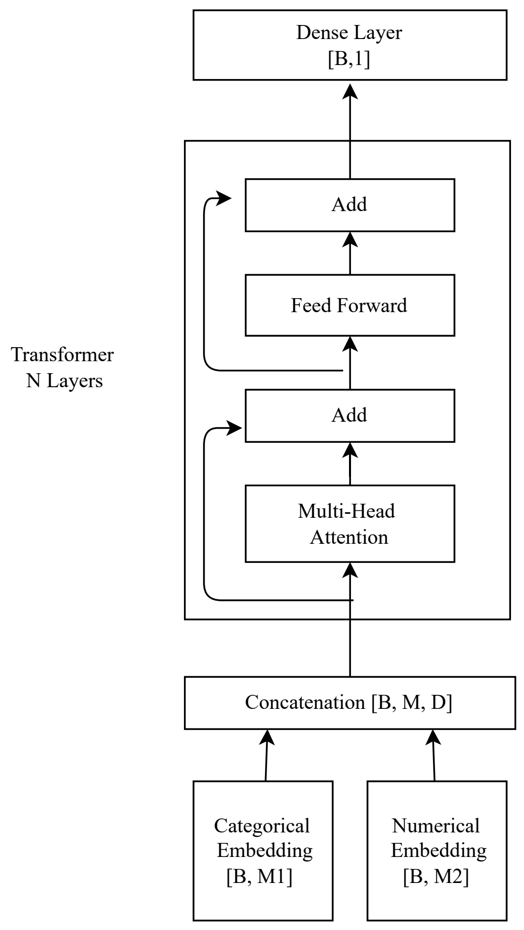
2.2 Neural Networks

We compare 3 basic architectures: Residual Multilayer Perceptions, Convolutional neural net, and Transformer based nets. Transformer models have the best performance and fastest convergence. Figure 4 shows the architecture of a modified Transformer. It contains column wise embedding layers for each variables, Transformer layers as encoder, and a dense layer for output prediction.

B denotes the number of observations, M_1 is number of categorical column, M_2 is the number of numerical columns, and $M = M_1 + M_2$. The embedding layer, a list of linear transformations, maps each independent variable into a D dimensional vector that are then passed into Transformer layers. The final dense layer includes two linear transformations: one for the dimension M that combines all independent variables into 1, and the other one maps the dimension D into a real number. Dropout layers are placed both in the feed forward layer and multi-head attention layer.

Narang et al. (2021) shows the vanilla Transformer is good enough and empirically shows Gated Linear Units (GEGLU) and Root Mean Square Norm (RMS Norm) outperform recent modifications. Huang et al. (2020) theoretically show proper model initialization can bound the variance in optimization up-

Figure 4: Transformer Architecture



date steps and thus eliminate the learning rate warn up and layer norms. We follow their initialization scheme and remove all layer norms in the Transformer layer. We also replace non linear activation by GEGLU in the feed forward block.

Hyper-parameters include number of Transformer Layer N, embedding dimension D, feed forward hidden dimension, multi-head dimension, batch size, weight decay, learning rate, and dropout.

We use Pytorch (Paszke et al. 2019) and Pytorch Lightning (Falcon et al. 2019) to build neural nets.

3 Experiments

We start with a small subset of data to test various model settings and narrow down parameter search space. We use 40% of data as training set, 10% as validation set, and 10% as test set. All models are trained on the same dataset and benchmarked in the same test set. The evaluation metric is Mean Square Error (MSE) on the log of income.

We find treating numeric variables as categorical yield better results for neural nets but worse for tree models. Tree models use standardized age, hours worked, and weeks worked last year as numeric variables. The error analysis suggest we get better performance by penalizing higher weights for college students and grouping ages separately for below 25 and over 75.

In the fine-tuning phrase, we split the data into 90% training and 10% validation. Models are early stopped by monitoring the validation set.

3.1 Parameter Search

In the tree models, we use “Autogluon” default parameter search space for XGBoost and CatBoost as they already control out of sample variance. But we add regulizations parameters for the LightGBM. They are “bagging_fraction”, “bagging_freq”, “lambda_l1”, “lambda_l2”. Parameter search is based on Gaussian process Bayesian optimization. Self-ensemble models are reported in table 1.

Table 1: Ensemble Trees in Test Set

Model	MSE
LightGBM	**0.5156**
XGBoost	0.5333
CatBoost	0.5325

In the neural nets, we use Adam for optimization and train models in mixed precision (FP16/FP32), resulting in faster convergence and lower GPU memory consumption. We fix batch size at 2048 as we find larger batches have no impact on test results. We first search optimal combinations on learning rate (LR), transformer layers (N), feed forward dimension (FF), and embedding (E). Then, we find dropout and weight decay for regularization.

Models are early stopped when there are no improvement in the validation set for 10 epochs or when gradient explode. All models are then benchmarked in the test set. Selected results are reported in table 2.

We find learning rate larger than 2e-3 guarantees gradient explosion, and 8e-4 is not stable. Deeper models seem to perform better than wider models, but they require longer training time and heavier regularizations. Wider models also bottleneck training in the multi-head attention blocks. We try Sparsemax to replace softmax in the attention block,

Table 2: Transformers in Test Set

LR	E	FF	N	Parameters	MSE
8e-4	128	512	4	1.442 M	NA
6e-4	128	512	4	1.442 M	0.5745
4e-4	128	512	4	1.442 M	0.5842
6e-4	64	256	4	0.393 M	0.6269
6e-4	256	1024	4	5.506 M	0.5578
6e-4	**128**	**512**	**6**	**2.100 M**	**0.5562**
6e-4	128	512	6	2.100 M	0.5591
4e-4	128	512	6	2.100 M	0.5641
6e-4	128	256	6	1.507 M	0.5865

Figure 5:

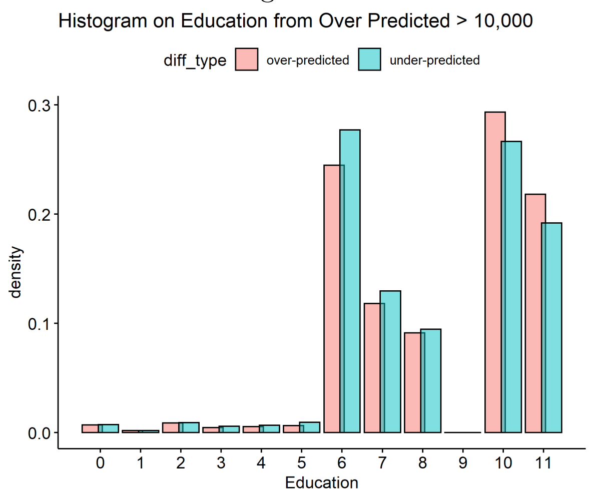
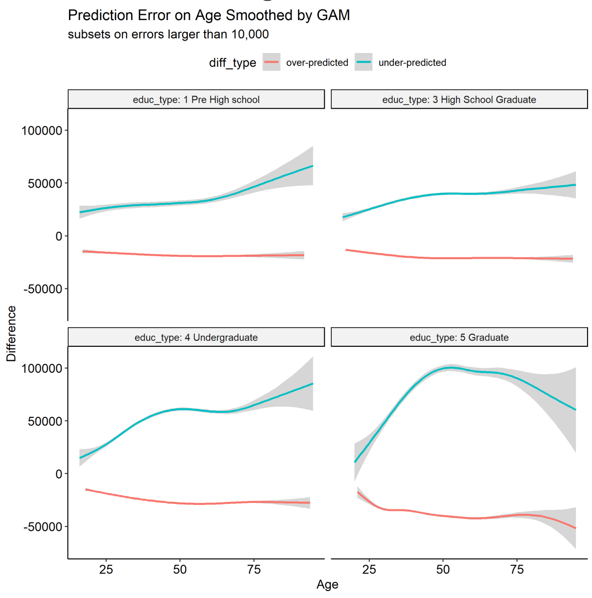


Figure 6:



and we find it benefits larger models as it creates extra regularization by zeroing out less useful features. However, the training time is much longer.

We decide to use 128 embedding size, 512 feed forward dimension, 6 Transformer layers, and 6e-4 learning rate. We find 5e-5 weight decay and 0.4 dropout have the best performance.

We also try to replace the dense layer by residual blocks, but models quickly overfit the data. We also try an image model by converting the embedding layer into an image-like structure, which is a 4 dimension tensor with batch size, number of variables, img width and img height, and then pass into the Efficient Net V2 (Tan and Le 2021). But performances are much worse than Transformers.

3.2 Error Analysis

Tree models report age, hours worked per week (uhrswork), and education are import features. We find age and education gives the highest unbalance prediction error.

The histogram in figure 5 shows the majority of large errors are from high school and college graduates. Figure 6 shows the prediction errors larger than 10,000. Young people and seniors have the largest error prediction, and graduates with more than 5 years of college study have the highest unbalanced distribution. We expect the recent graduates have lower income in their early careers.

We re-run both models with different feature settings. In tree models, pre college observations have weight of 1, colleges with 1-4 years of study have weight of 2, and graduates have weight of 3. We compare auto weight search, manual settings, and equal weights. In neural nets, we add the same weights in the loss function. We also group ages below or equal to 25 as 1 group and ages above or equal to 75 as another group. Results are listed in table 3. The feature engineering is effective.

3.3 Fine Tuning

We train the two models on the full sample. Hyperparameter search space for LightGBM is narrowed according to previous experiments and increase search iterations. Neural nets learning rates are adjusted lower by multiplying 0.8 when there are no improvement for 5 epoch. Model is early stopped when there are no improvements for 20 epochs in the validation set. To build a mixed ensemble mode, we try a simple weighted average on the two. The opti-

Table 3: Feature Engineering in Test Set

Model	Penalty & Grouping	MSE
LightGBM	False	0.5156
LightGBM	**True**	**0.5010**
Transformer	False	0.5562
Transformer	**True**	**0.5312**

mal weight for LightGBM is 0.999, meaning Transformer has no usage. We also use the predictions as features to train ensemble trees in LightGBM that gives slightly better performance.

Table 4 shows the MSE for the whole training set. The final prediction on the test set is round(exp(log(*incwage*))) as it has around 3e-8 less error than rounding down or up. We find adding magics yields much better performance, so we use it for the final submission.

Table 4: Full Training Sample

Model	MSE
LightGBM	0.3901
Transformer	0.4265
Mixed Ensemble (Weighted)	0.3990
Mixed Ensemble (Trees)	**0.3873**
Magics	??

References

Chen, Tianqi and Carlos Guestrin (Aug. 2016). "XGBoost". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. DOI: `10.1145/2939672.2939785`. URL: `http://dx.doi.org/10.1145/2939672.2939785`.

Dorogush, Anna Veronika, Vasily Ershov, and Andrey Gulin (2018). *CatBoost: gradient boosting with categorical features support*. arXiv: `1810.11363 [cs.LG]`.

Erickson, Nick et al. (Mar. 2020). "AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data". In: *arXiv:2003.06505 [cs, stat]*. arXiv: 2003.06505. URL: `http://arxiv.org/abs/2003.06505`.

Falcon, William et al. (2019). "Pytorch lightning". In: *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning* 3, p. 6.

Huang, Xiao Shi et al. (Nov. 2020). "Improving Transformer Optimization Through Better Initialization". en. In: *Proceedings of the 37th International Conference on Machine Learning*. PMLR, pp. 4475–4483. URL: `https://proceedings.mlr.press/v119/huang20f.html`.

Ke, Guolin et al. (2017). "Lightgbm: A highly efficient gradient boosting decision tree". In: *Advances in neural information processing systems* 30, pp. 3146–3154.

Narang, Sharan et al. (2021). "Do Transformer Modifications Transfer Across Implementations and Applications?" In: arXiv: `2102.11972 [cs.LG]`.

Paszke, Adam et al. (Dec. 2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *arXiv:1912.01703 [cs, stat]*. arXiv: 1912.01703. URL: `http://arxiv.org/abs/1912.01703`.

Ruggles, Steven et al. (2021). *IPUMS USA: Version 11.0 [dataset]*. URL: `https://doi.org/10.18128/D010.V11.0`.

Tan, Mingxing and Quoc V. Le (June 2021). "EfficientNetV2: Smaller Models and Faster Training". In: *arXiv:2104.00298 [cs]*. arXiv: 2104.00298. URL: `http://arxiv.org/abs/2104.00298`.

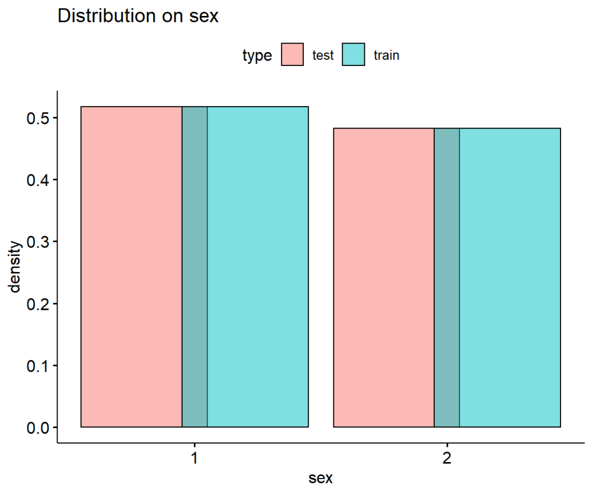
A Additional Plots on Preliminary Data Analysis

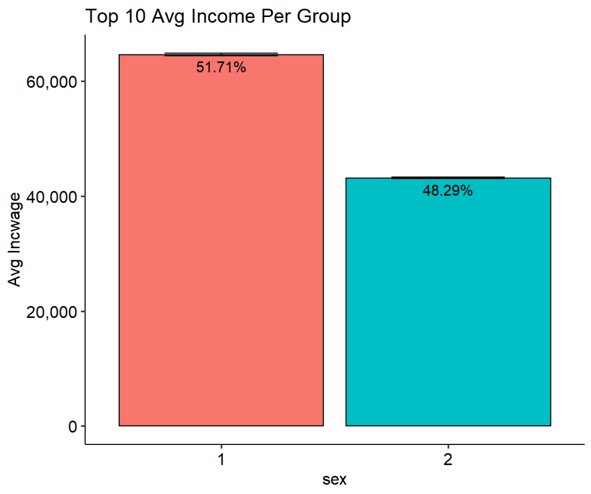
There are almost no difference in the train and test samples. Neither ANOVA nor Kruskal-Wallis test reject the null hypothesis that the two dataset have different distribution in any independent variables.

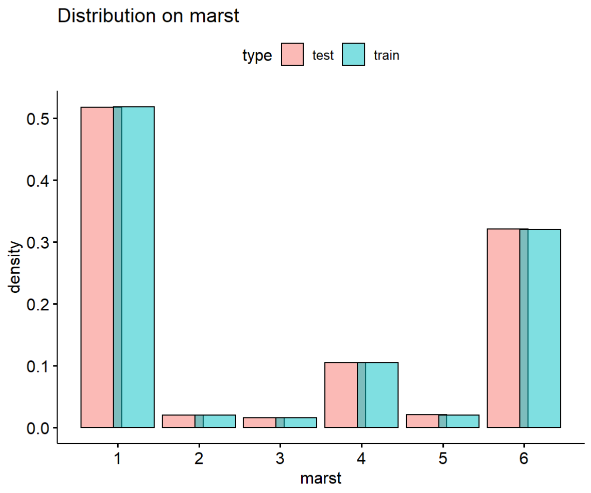
If we only look at single variable against income, we have the following observations:

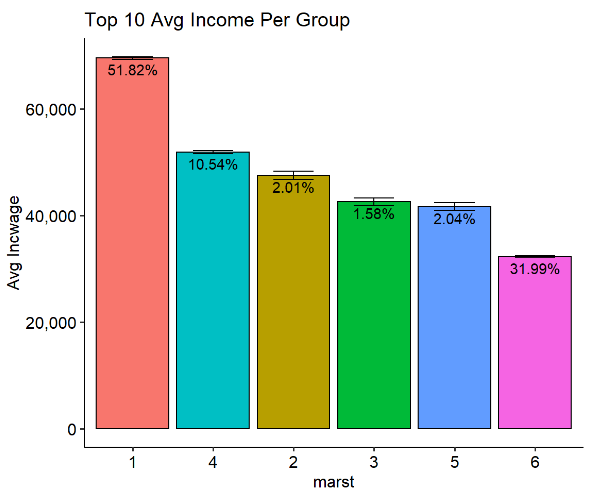
- male(sex=1) has slightly higher income than female (sex=2);
- married (marst=1) has the most observation and highest income; but single (marst=6) has the second most observation and lowest income;
- white (race=1) has the most observation but income ranks at 4th; Chinese, Japanese, other asian (race=4,5,6) have less observations but highest income;
- EU member counties (bpl in 400 range) have the highest income
- minority language speakers have the highest average income
- 5 years + college (educ=5) and 4 years + (educ=4) are the top earners.
- Engineering, Transportation Sciences and Technologies, Biology and Life Sciences (degfield = 24, 59, 36) have the highest avg income.

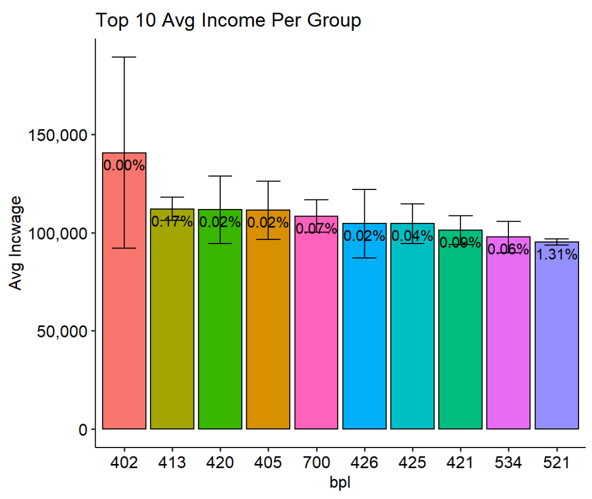
If we use embedding for each category variables, degfield shall get special attention, followed by educ as they seem to have high predictive power. language and bpl may need to consider sparse setting as the majority speakers are English speakers born within the U.S., but they have diverse income.

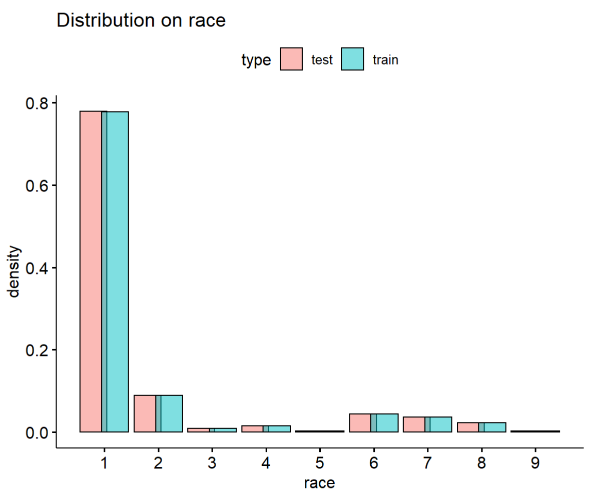


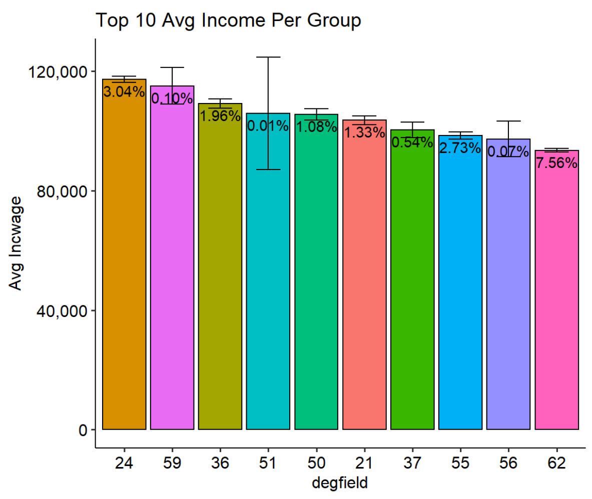


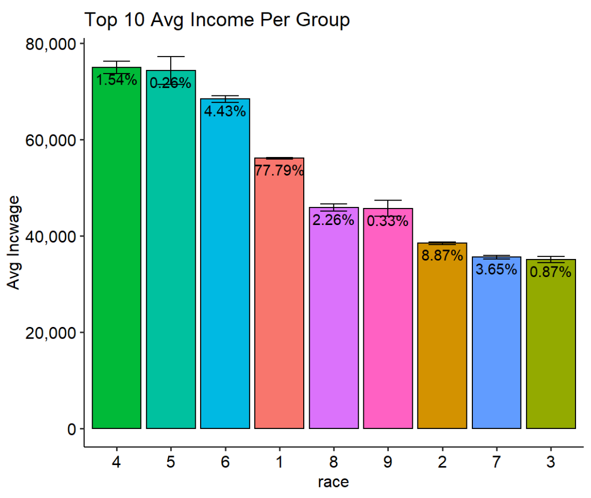


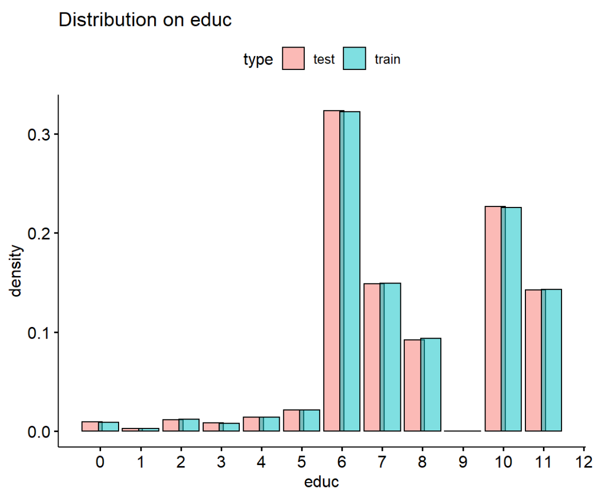

Top 10 Avg Income Per Group
Avg Incwage
marst
51.82%
10.54%
2.01%
1.58%
2.04%
31.99%

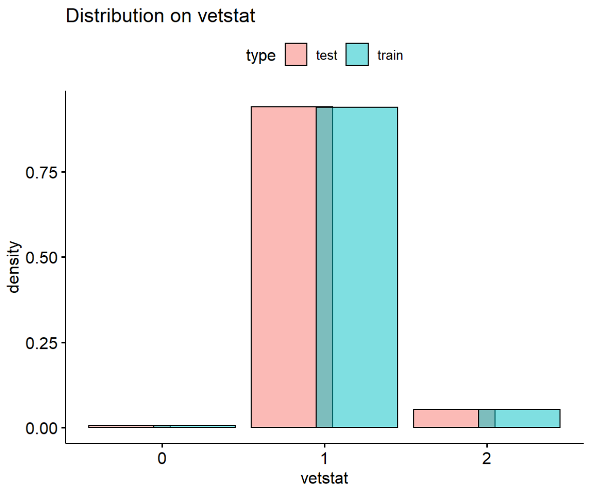

Top 10 Avg Income Per Group
Avg Incwage
bpl
0.00%
0.17%
0.02%
0.02%
0.07%
0.02%
0.04%
0.09%
0.06%
1.31%

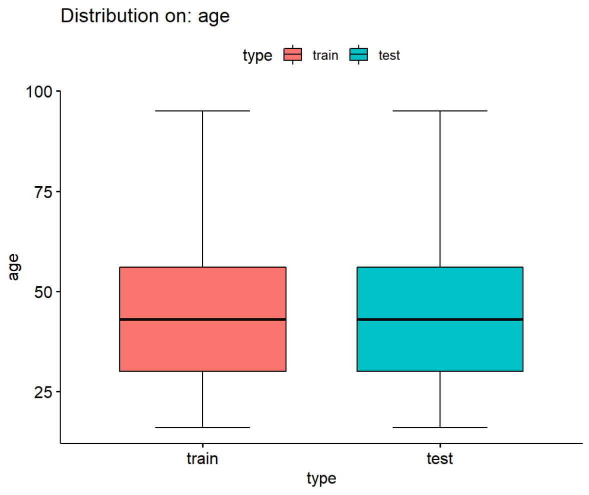

Distribution on race
type
test
train
density
race

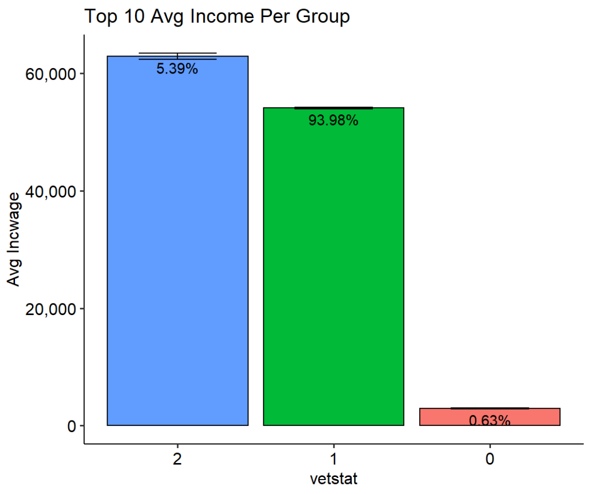

Top 10 Avg Income Per Group
Avg Incwage
degfield
3.04%
0.10%
1.96%
0.01%
1.08%
1.33%
0.54%
2.73%
0.07%
7.56%

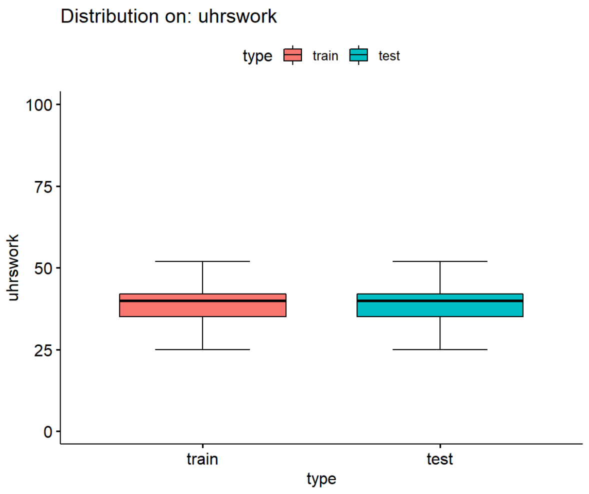

Top 10 Avg Income Per Group
Avg Incwage
race
1.54%
0.26%
4.43%
77.79%
2.26%
0.33%
8.87%
3.65%
0.87%

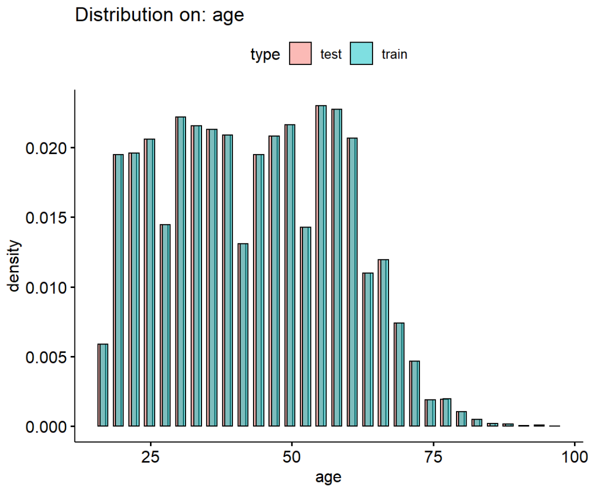

Distribution on educ
type
test
train
density
educ

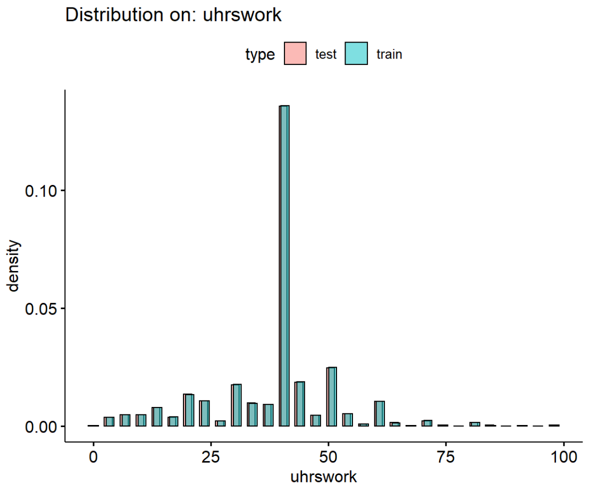

Distribution on vetstat
type
test
train
density
0.75
0.50
0.25
0.00
0
1
2
vetstat


Distribution on: age
type
train
test
age
100
75
50
25
train
test
type


Top 10 Avg Income Per Group
Avg Incwage
60,000
40,000
20,000
0
5.39%
93.98%
0.63%
2
1
0
vetstat


Distribution on: uhrswork
type
train
test
uhrswork
100
75
50
25
0
train
test
type


Distribution on: age
type
test
train
density
0.020
0.015
0.010
0.005
0.000
25
50
75
100
age


Distribution on: uhrswork
type
test
train
density
0.10
0.05
0.00
0
25
50
75
100
uhrswork