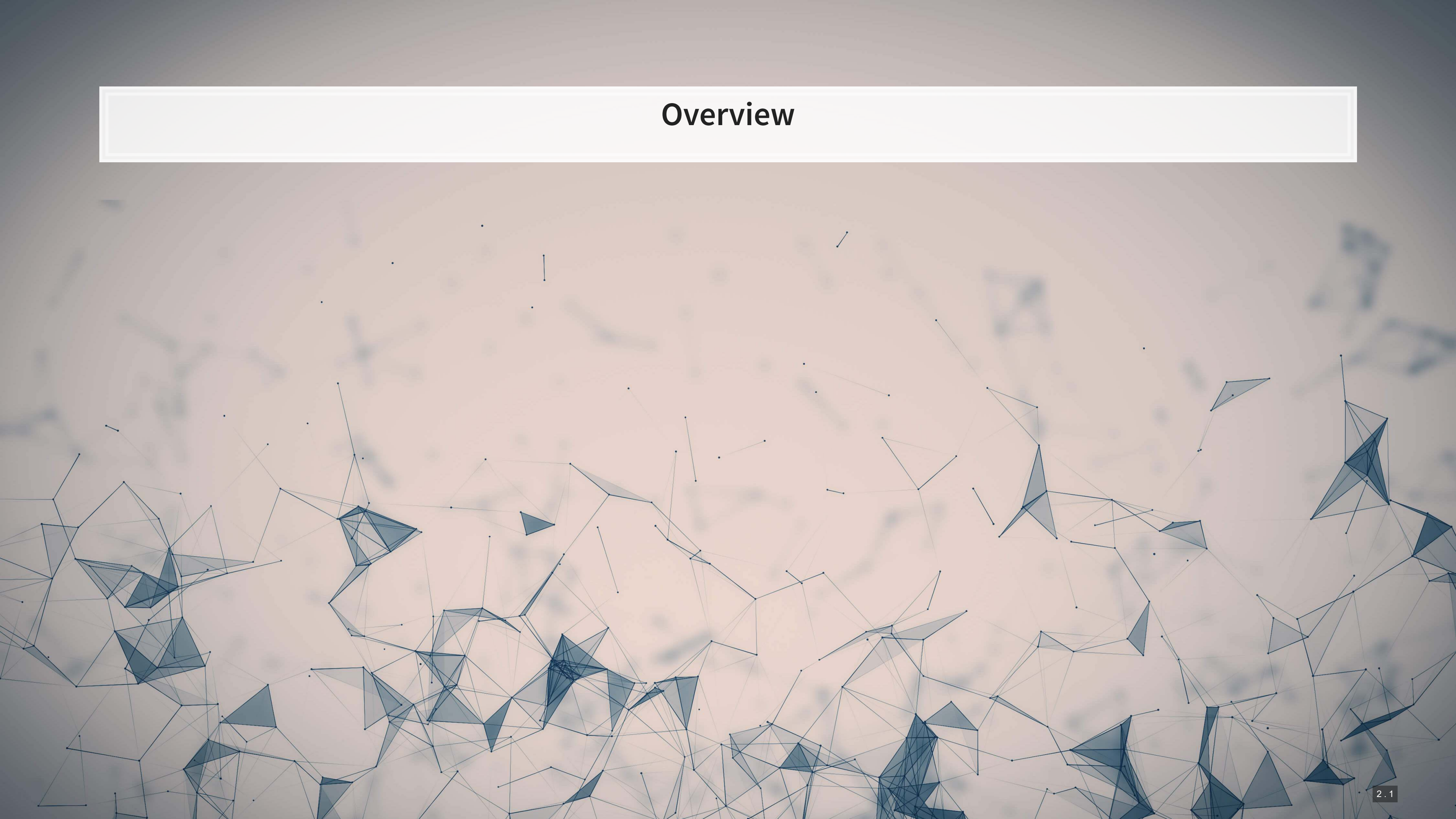


ML for SS: Bias

Session 10

Dr. Richard M. Crowley
rcrowley@smu.edu.sg
<http://rmc.link/>

Overview



Papers

- Wich, Bauer and Groh (2020)
 - A paper using SHAP to understand an impact of political bias
- Lundberg et al. (2018)
 - A practical use of SHAP for model explainability
 - The team behind this paper contains the team from the original SHAP paper (Lundberg and Lee (2017))
- Rambachan et al. (2020)
 - Discusses algorithmic fairness and sources of bias in algorithms

Technical Discussion

Focus on the SHAP method

Python

- Use the `shap` library
 - By the original author team
 - Great visualization support
 - Decent documentation
 - Has some bugs
 - Sometimes you need to use older packages with it

R

- For XGBoost, you can use `SHAPforxgboost`
- For accessing the python package in R, use `shapper`
- For native SHAP, use `shapr`, but it is missing a lot of features

Python's support is a lot better here unless your model is an XGBoost

SHAP

What exactly is SHAP?

Aims to provide an explanation of the importance of model inputs in explaining model output

- Game theoretic and theory driven
- Unifies six other methods that tried to address this problem
- It is a model itself: a model to explain models
- Provides a simple to understand output

SHAP: *SH*apley *Ad*ditive *exP*lanations

- Based on Shapley, 1953, “A value for n-person games.”
- SHAP itself is from Lundberg and Lee (2017)

Principles of SHAP

1. Local accuracy

- The simple model is able to accurately predict a model output on small subsets of the data

2. Missingness

- SHAP only uses data the original model had access to
- If data was missing from the original model, SHAP won't use it

3. Consistency

- Akin to transitivity conditions in utility theory (Savage Axioms)
 - But instead of “utility,” we have “simplified model's input's contribution”

SHAP in more detail

SHAP is, per Lundberg and Lee (2017), the unique solution that maintains local accuracy and consistent from a class of methods called *additive feature attribution methods* (AFAM)

AFAM's have a linear function of binary variables where $z' \in \{0, 1\}^M$ where M is a number of simplified input features, $\phi_i \in \mathbb{R}$ is the feature importance, $x = h_x(x')$, and $g(z') \approx f(h_x(z'))$ when $z' \approx x'$.

$$g(z') = \phi_0 + \sum_{i=1}^M \phi_i z'_i$$

- 6 other methods in the literature also fit in the class
 - LIME, DeepLIFT, Layer-Wise Relevance Propagation, Shapley regression values, Shapley sampling values, Quantitative input influence
 - These methods were approximating SHAP

SHAP: Local accuracy

$$f(x) = g(x') = \phi_0 + \sum_{i=1}^M \phi_i x'_i$$

- $g(x')$ is the explanation model of $f(x)$ where $x = h_x(x')$ and $\phi_0 = f(h_x(0))$

Not all other methods have this

SHAP: Missingness

$$x'_i = 0 \Rightarrow \phi_i = 0$$

- “Features missing in the original input [have] no impact”

All AFAM models have this

SHAP: Consistency

Let $f_x(z') = f(h_x(z'))$ and $z' \setminus i$ denote setting $z'_i = 0$. For any two models f and f' :

$$f'_x(z') - f'_x(z' \setminus i) \geq f_x(z') - f_x(z' \setminus i) \quad \forall z' \in \{0, 1\}^M \Rightarrow \phi_i(f', x) \geq \phi_i(f, x)$$

- Recall that ϕ is measuring feature importance of i
- If removing i drops the prediction more under f'_x than under f_x , then it has more feature importance under f'_x than under f_x

Not all other methods have this

SHAP: The solution

$$\phi_i(f, x) = \sum_{z' \subseteq x'} \frac{|z'|!(M - |z'| - 1)!}{M!} [f_x(z') - f_x(z' \setminus i)]$$

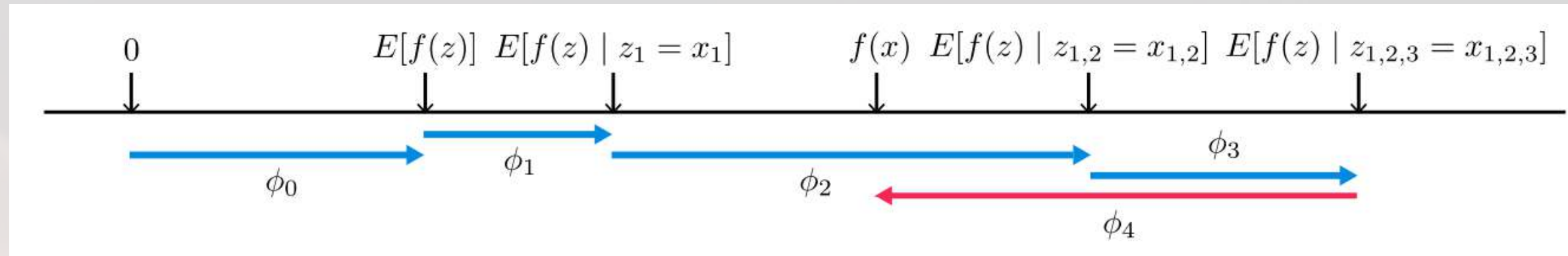
- Where:
 - $|z'|$ is the number of non-zero entries in z'
 - $z' \subseteq x'$ is the set of all z' s.t. the non-zero entries are a subset of the non-zero entries in x'

Combinatoric weighting to the difference element i adds to f_x

SHAP sets $f_x(z') = f(h_x(z')) = (E)[f(z)|z_S]$; S is the set of non-zero indexes in z'

Then approximate it all

Intuition of SHAP



- SHAP is defined by a series of [conditional] expectations of the impact of an input
- For linear models, order of selecting inputs has no effect
- For nonlinear models, SHAP averages inputs' conditional expected impact over all possible orderings
 - This is computationally intensive on high-dimensional data

Prepping SHAP

An example of quantifying bias

- Data: City of Chicago salaries
 - 33,586 employees
- Trained using a simple XGBoost model
- Features:
 - Job title
 - Department
 - Full time / part time
 - Salaried or hourly
 - Female

Is there gender bias in annual compensation?

The data

```
vars = ['Job.Titles', 'Department', 'Full.Time', 'Salaried', 'Female']  
df[vars]
```



```
##           Job.Titles      Department  Full.Time  \  
## 0           SERGEANT           POLICE           1  
## 1  POLICE OFFICER (ASSIGNED AS DETECTIVE)  POLICE           1  
## 2           Other  GENERAL SERVICES           1  
## 3           Other  WATER MGMNT           1  
## 4           Other  TRANSPORTN           1  
## ...           ...           ...           ...  
## 33581  POLICE OFFICER           POLICE           1  
## 33582  POLICE OFFICER           POLICE           1  
## 33583  POLICE OFFICER           POLICE           1  
## 33584  POLICE OFFICER           POLICE           1  
## 33585           Other           Other           1  
##  
##           Salaried  Female  
## 0           1      0.0  
## 1           1      1.0  
## 2           1      1.0  
## 3           1      0.0  
## 4           0      0.0  
## ...           ...      ...  
## 33581           1      1.0
```


One hot encoding categorical data

- Pandas has a function for this, `pd.get_dummies()`
 - `prefix=` lets us name the columns of the output
- As `pd.get_dummies()` outputs a new data frame only containing the new columns, we need to join them back
 - `df.join()` makes this quick and easy

```
one_hot1 = pd.get_dummies(df['Job.Titles'], prefix='Job.Titles')
one_hot2 = pd.get_dummies(df['Department'], prefix='Department')

df = df.join(one_hot1)
df = df.join(one_hot2)
```



Prepping XGBoost

We did this in Session 3

```
vars = one_hot1.columns.tolist() + \  
       one_hot2.columns.tolist() + \  
       ['Full.Time', 'Salaried', 'Female']  
dtrain = xgb.DMatrix(df[vars], label=df['Salary'], feature_names=vars)
```

```
param = {  
    'booster': 'gbtree',           # default -- tree based  
    'nthread': 8,                 # number of threads to use for parallel processing  
    'objective': 'reg:squarederror', # RMSE error  
    'eval_metric': 'rmse',        # maximize ROC AUC  
    'eta': 0.3,                   # shrinkage; [0, 1], default 0.3  
    'max_depth': 6,               # maximum depth of each tree; default 6  
    'gamma': 0,                   # set above 0 to prune trees, [0, inf], default 0  
    'min_child_weight': 1,        # higher leads to more pruning of tress, [0, inf], default 1  
    'subsample': 1,               # Randomly subsample rows if in (0, 1), default 1  
}  
num_round=30
```

Building our model and prepping SHAP

- We call `xgb.train()` to fit our XGBoost model

```
model_xgb = xgb.train(param, dtrain, num_round)
```



- Since XGBoost is a tree-based model, we will use SHAP's `shap.TreeExplainer()` function to analyze the model
- Since we only have in-sample data, we will compute SHAP on the same data the XGBoost model was fit to
- We will also prepare a small sample for more CPU-intensive analyses

```
explainer = shap.TreeExplainer(model_xgb)
shap_values = explainer(df[vars])

df_small = df.sample(frac=0.01)
shap_values_small = explainer(df[vars])
```



Explaining a single observation

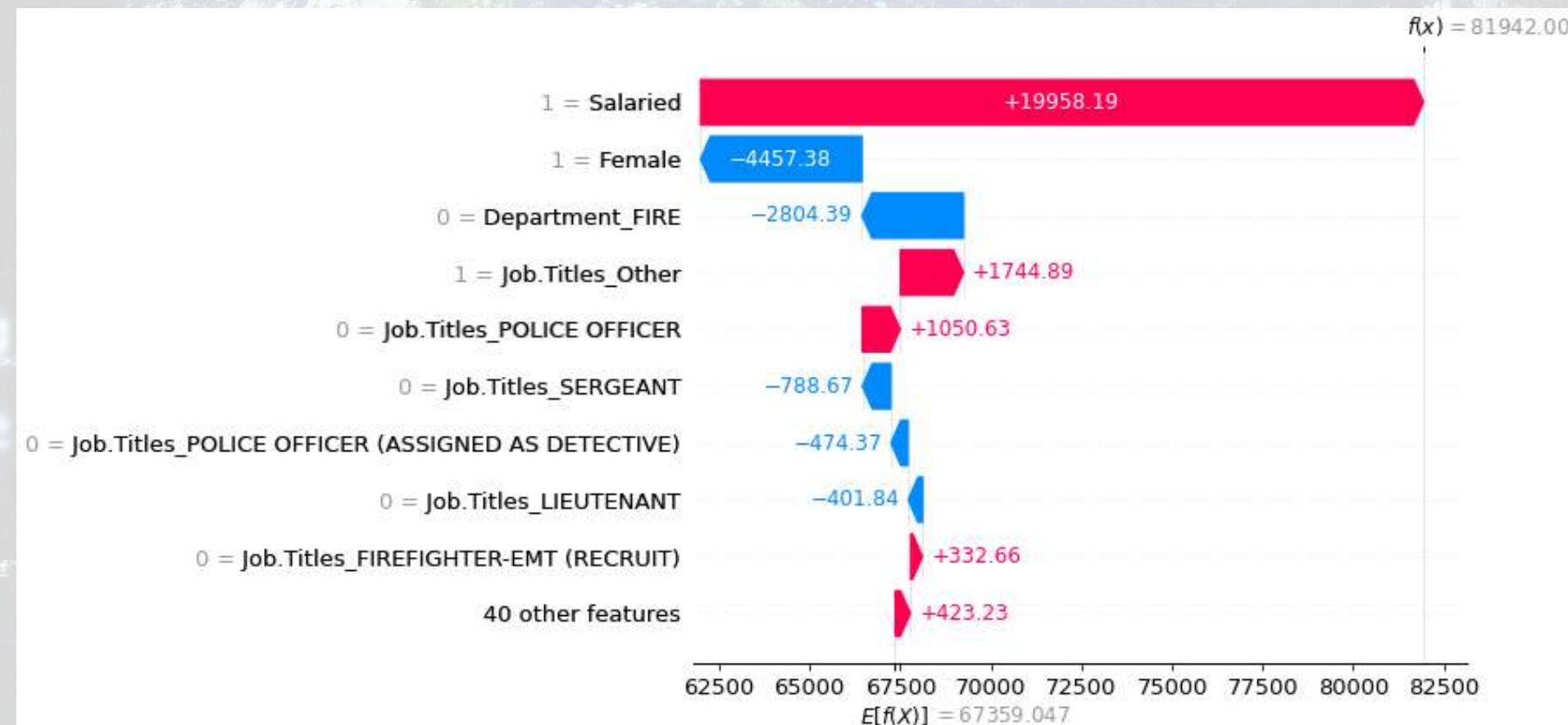
```
shap.plots.waterfall(shap_values[0])
```



Here we see that having `Female=0` was the fourth most influential feature in the model, and that it led to a *higher* predicted salary

Explaining a single observation

```
shap.plots.waterfall(shap_values[2])
```



Here we see that having `Female=1` was the second most influential feature in the model, and that it led to a *lower* predicted salary

Charting with SHAP

A more concise point visualization

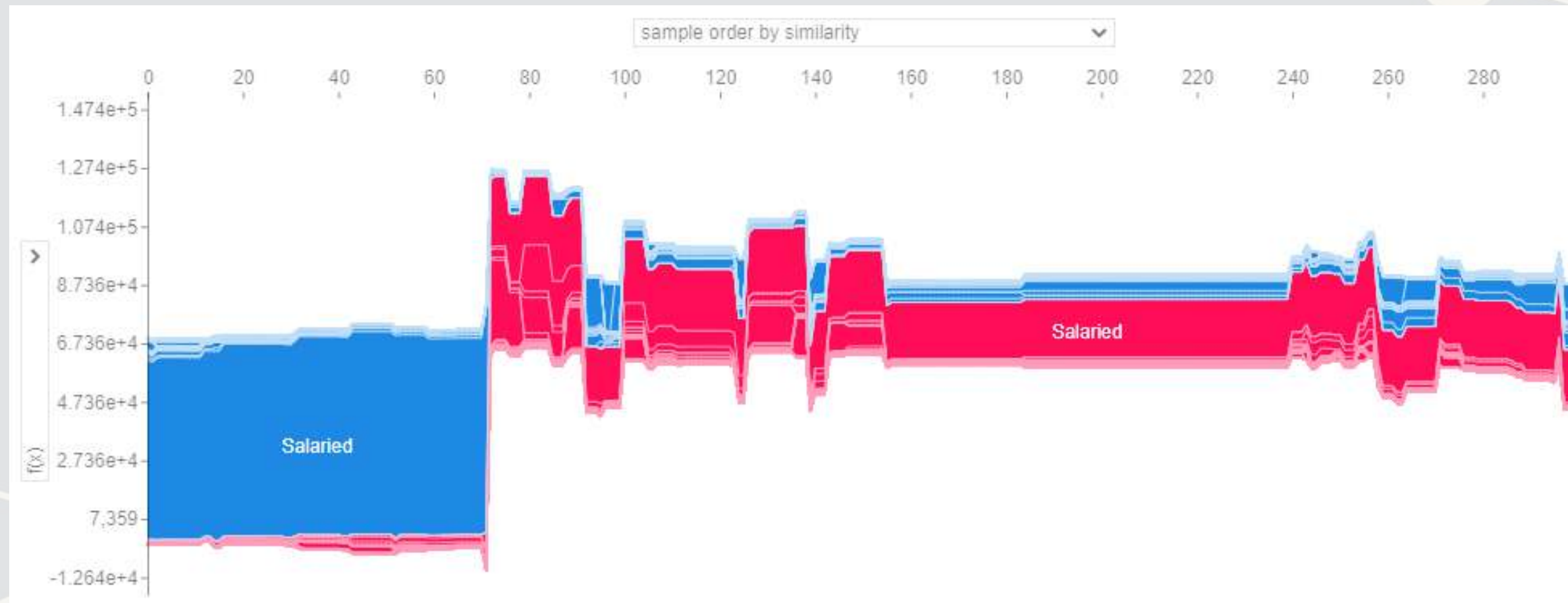
```
shap.plots.force(shap_values[1])
```



Aggregating across the data

N=300

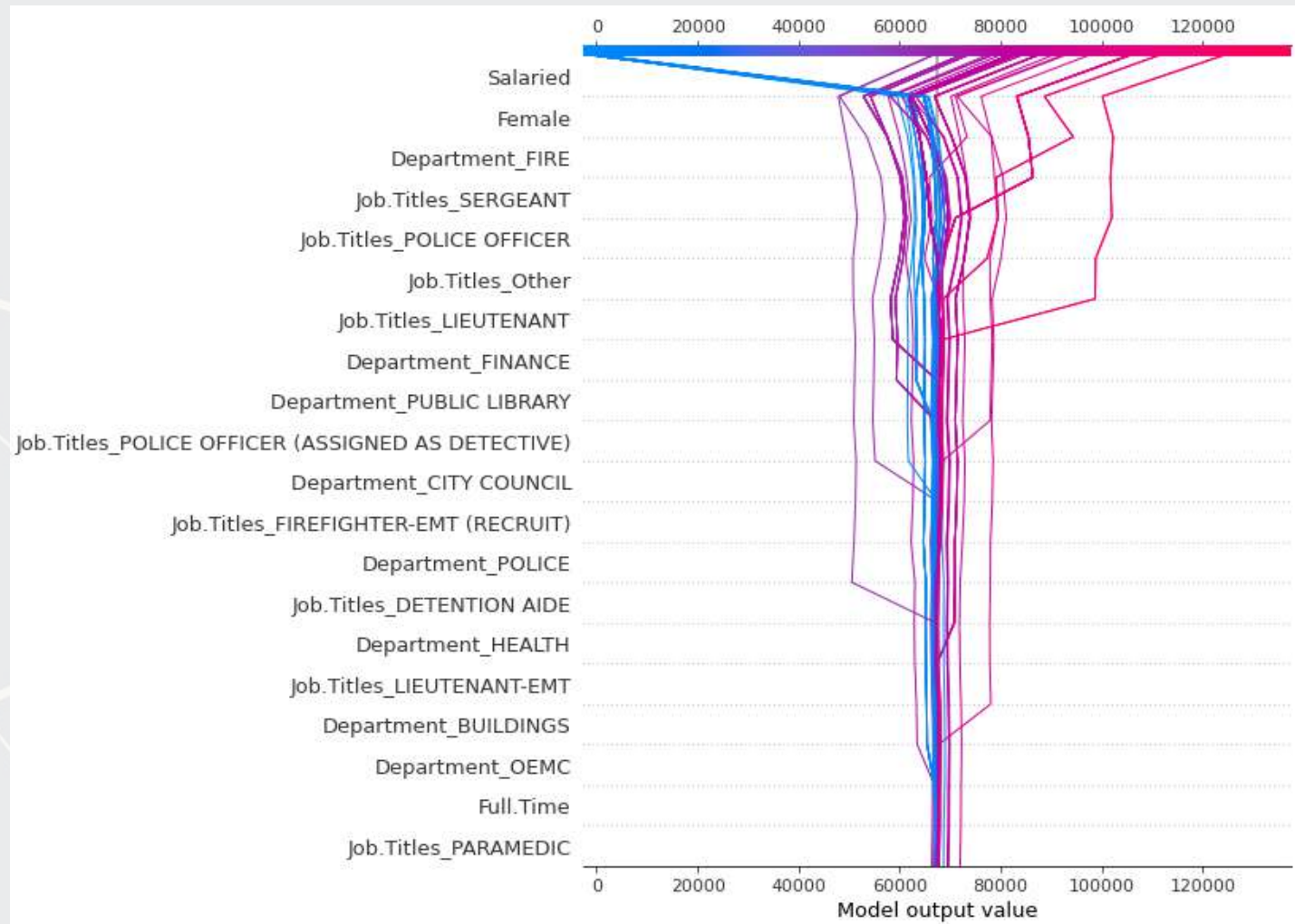
```
shap.plots.force(explainer.expected_value, shap_values.sample(N).values, feature_names=vars)
```



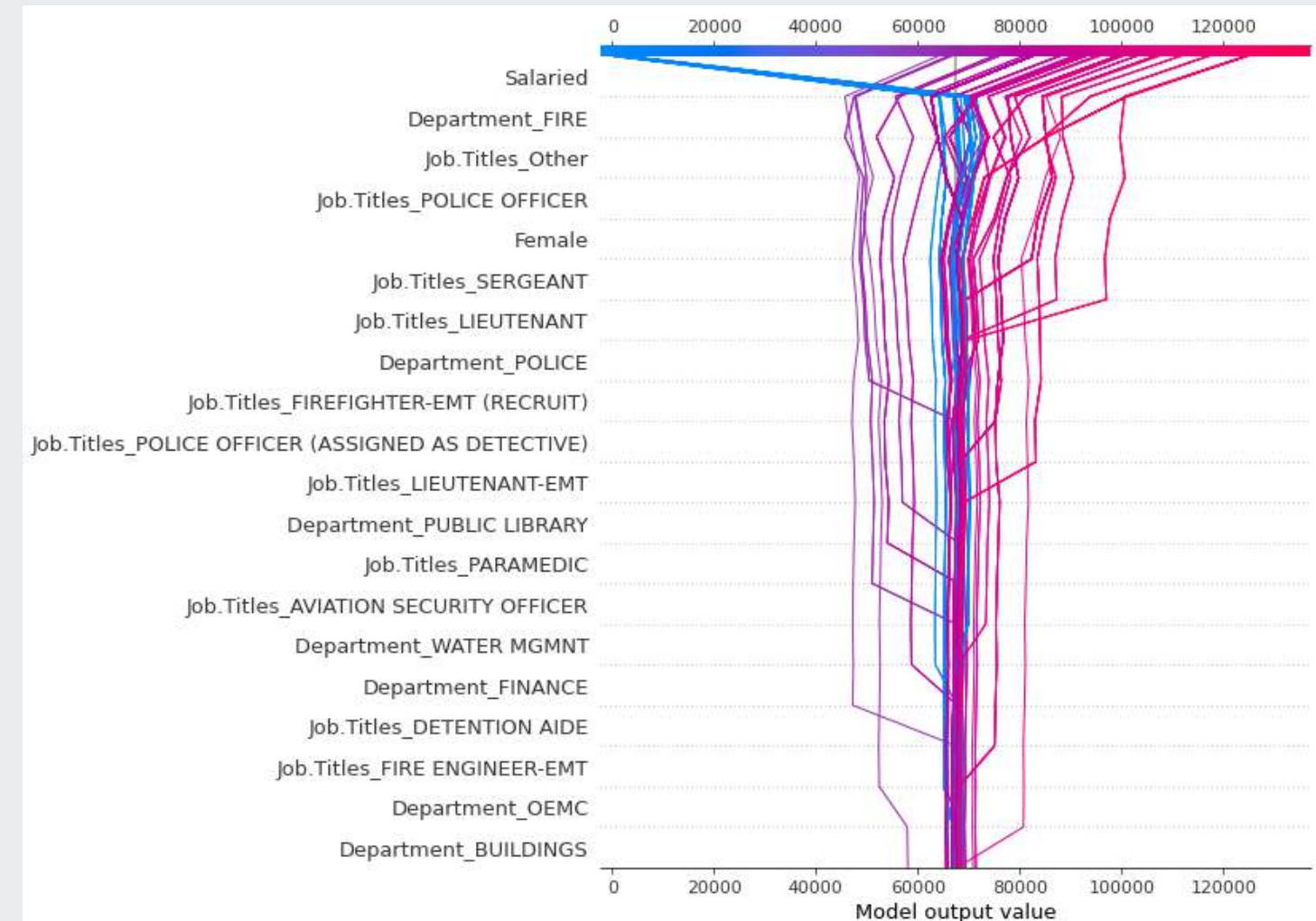
Seeing more variables' impact

- A “Decision plot” uses a line chart to show the impact of more measures across the data

```
shap.decision_plot(  
    explainer.expected_value,  
    explainer.shap_values(df_small[df_small.Female==1][vars  
    feature_names=vars])
```



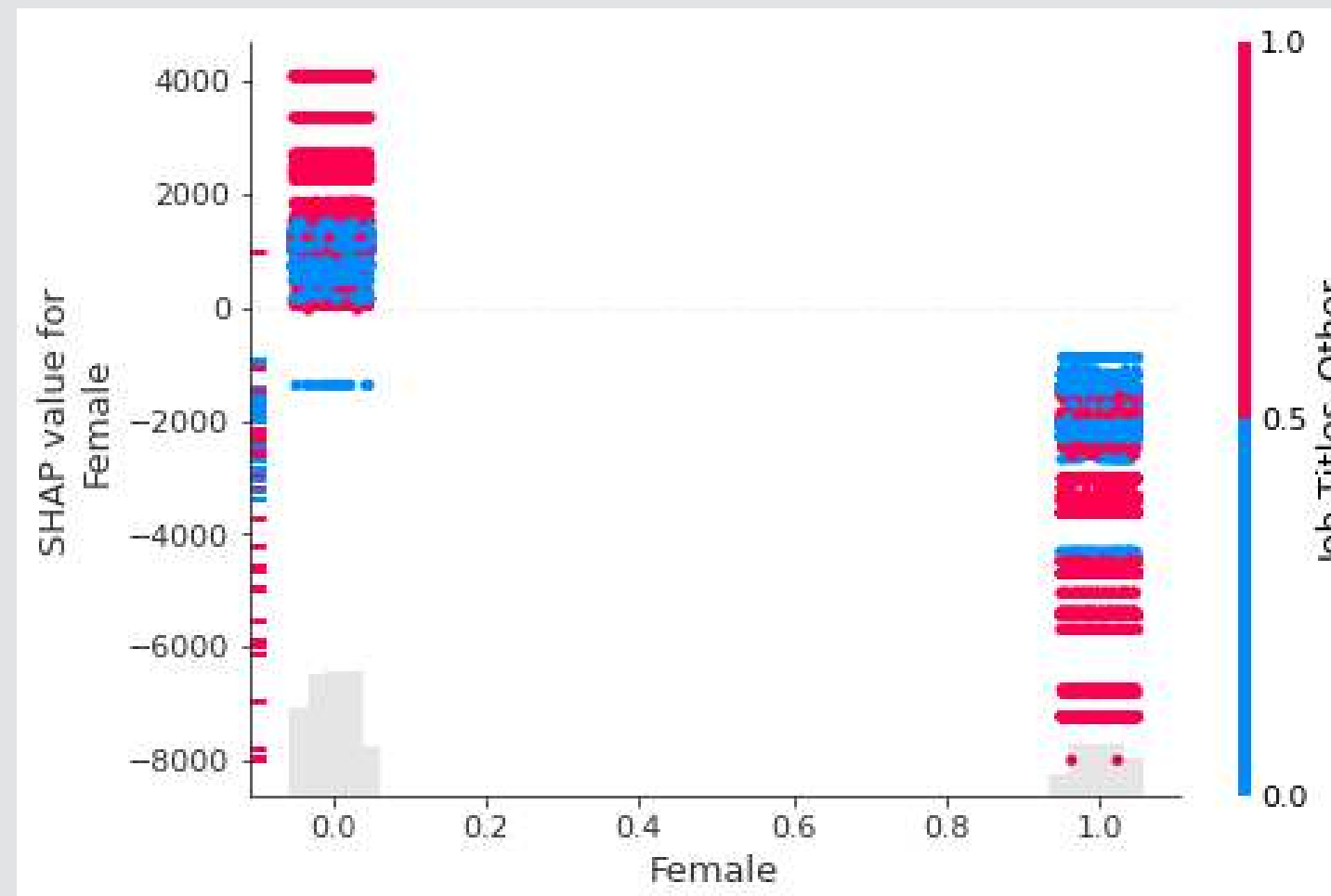
```
shap.decision_plot(  
    explainer.expected_value,  
    explainer.shap_values(df_small[df_small.Female==0][vars  
    feature_names=vars])
```



Aggregate analysis of an individual variable

- If we want to see the full impact of “Female” on outcomes in our data, a scatter plot is useful

```
shap.plots.scatter(shap_values[:, "Female"], color=shap_values)
```

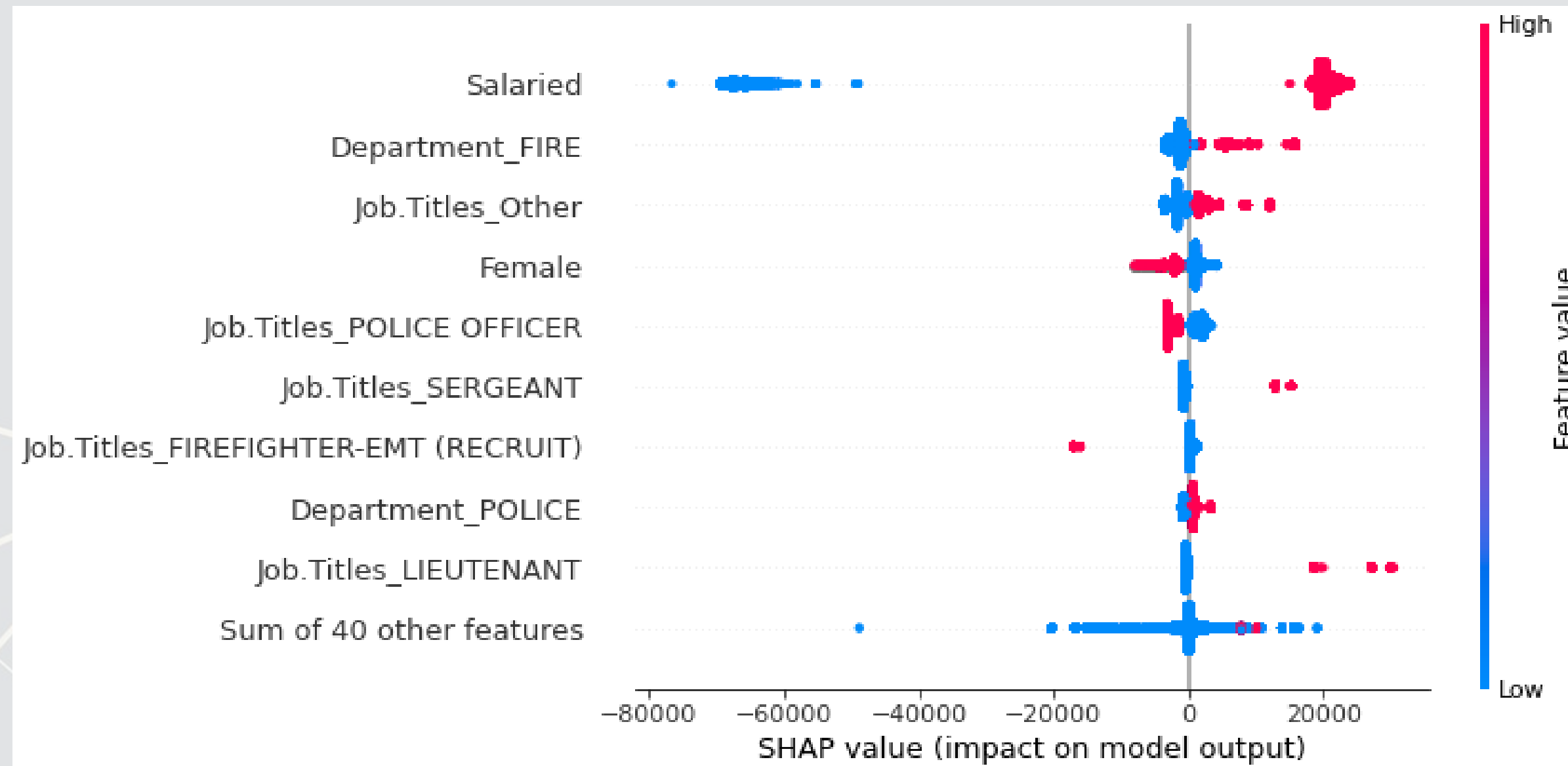


Remember that our model is nonparametric! Signs can be different even when the variable doesn't change due to interactive effects

Multiple scatterplots at once: Bee swarm

- If you want a concise way to present multiple variables, the bee swarm plot can be useful

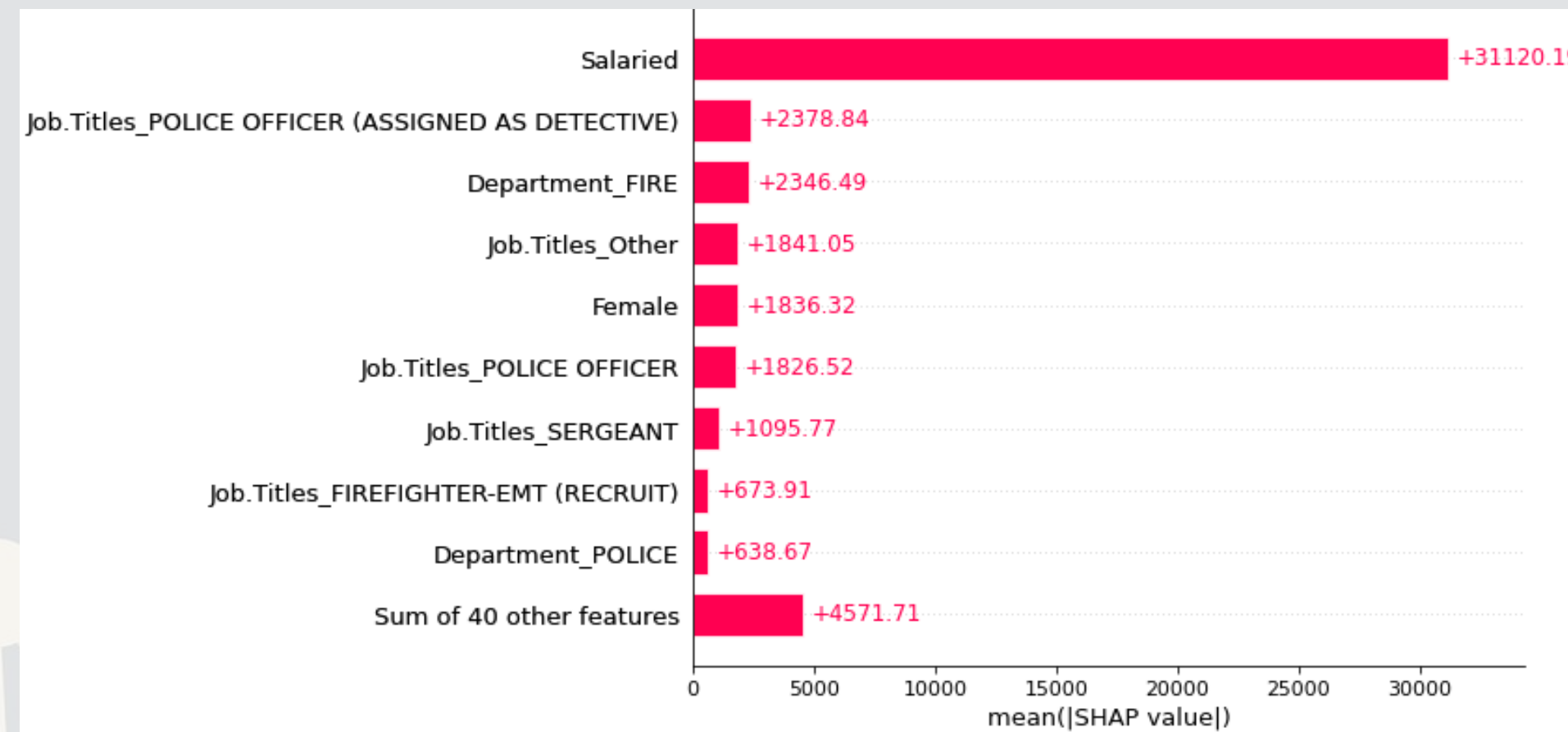
```
shap.plots.beeswarm(shap_values)
```



Importance plot

- Lastly, we can replicate XGBoost's importance plot using |SHAP|

```
shap.plots.bar(shap_values)
```



This may not be useful for XGBoost since it already has an importance metric, but many other models lack it

Addendum: Using R

- If you are working explicitly with XGBoost, there is a great `SHAPforxgboost` package
- To interface with the python `shap` package, you can use `shapper`
- There is also `shapr`, though it isn't as full-featured.

Conclusion



Wrap-up

SHAP can provide some insight into models at the observation, group, and sample level

- For more complex models, this helps to unwrap the “black box” some

SHAP can provide us with [conditional] marginal effects-like analysis for more complex models

Packages used for these slides

Python

- numpy
- pandas
- shap
- xgboost

References

- Lundberg, Scott, and Su-In Lee. “A unified approach to interpreting model predictions.” In Proceedings of the 31st Conference on Neural Information Processing Systems. (2017).
- Lundberg, Scott M., Bala Nair, Monica S. Vavilala, Mayumi Horibe, Michael J. Eisses, Trevor Adams, David E. Liston et al. “Explainable machine-learning predictions for the prevention of hypoxaemia during surgery.” *Nature biomedical engineering* 2, no. 10 (2018): 749-760.
- Rambachan, Ashesh, Jon Kleinberg, Jens Ludwig, and Sendhil Mullainathan. “An economic perspective on algorithmic fairness.” In *AEA Papers and Proceedings*, vol. 110, pp. 91-95. 2020.
- Wich, Maximilian, Jan Bauer, and Georg Groh. “Impact of politically biased data on hate speech classification.” In *Proceedings of the Fourth Workshop on Online Abuse and Harms*, pp. 54-64. 2020.