In [1]:
```python
## IMPORTS
import importlib
import os
from pathlib import Path
import time

import ipywidgets as widgets
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

import IPython
import lib.utils as _utils
import lib.atus_tools as att
from lib.utils import get_project_logger, config_project_logger, \
    SUCCESS, rotate_ax_labels, get_color_cycle_list, get_x_array_for_barplots

# -------------
_LOGGER = get_project_logger().getChild("ipynb")
# importlib.reload(att)


# %matplotlib widget
# mpl.rcParams["pdf.fonttype"] = 42  # Make text editable in exported pdfs
```

```
[atus] DEBUG: `rtrend` Logger configured.
```

In [2]:
```python
# PARAMETERS
# =========================================

# Preprocessing directory path
# ----------------------------
# --- Baseline
# preproc_dir = Path("outputs/baseline/by_occupation/")
# preproc_dir = Path("outputs/baseline/by_industry/")
preproc_dir = Path("outputs/baseline/with_bootstrap_100/")

# --- Alternative/sensitivity
# preproc_dir = Path("outputs/alternative/income_classes_20-80_occ/")
# preproc_dir = Path("outputs/alternative/income_classes_20-80_ind/")

# --- Devtests
# preproc_dir = Path("outputs/tests/preproc_dev/")
## comment test
# Mosquito diel activity
# ----------------------
mosq_count_fname = Path("mosquito_diel_data/mosqdiel_general_counts.csv")
```

In [3]:
```python
def count_folders(directory):
    # Initialize a counter for folders
    folder_count = 0

    # Iterate over each item in the directory
    for item in os.listdir(directory):
        # Check if the item is a folder
        if os.path.isdir(os.path.join(directory, item)):
            # If it's a folder, increment the counter
            folder_count += 1

    return folder_count
```

In [4]:
```
hourly = False
# preproc_dir, hourly = Path("outputs/tests/main_bootstrap_test/"), False
preproc_dir, hourly = preproc_dir, True


# aggr_features = ["all", "income_id"]
# ------------
aggr_features=[
        "all", "income_id", "TUMONTH", "PESEX", "race_ethnicity",
        "is_outdoor_job",  "is_weekend", "job_and_weekend", "income_and_weekend"
#         "occupation_exposure_id",
    ]

# IMPORT PREPROCESSED DATA
# ========================
importlib.reload(att)

# ---

env = att.get_default_atus_env()

# === Load mosquito diel activity
mosq_diel_df = pd.read_csv(mosq_count_fname, index_col=[0, 1])
mosq_diel_df.columns.name = "hour"

mosq_diel_df #hourly mosquito count for different locations

env = att.get_default_atus_env()

# Load original ensemble
main_pre = att.ATUSPreprocBunch.from_dir(
    preproc_dir, aggr_features=aggr_features,
    import_raw=False,
    raise_on_not_found=False,
    tseries_is_hourly=hourly,

)
```

In [6]:
```
main_pre.feat_hist_df_dict['job_and_weekend']
```

Out[6]:

| exp_id | state_name | is_outdoor_job | is_weekend | 0 | 1 | 2 | 3 |
|---|---|---|---|---|---|---|---|
| 0 | Alabama | 0 | False | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| | | | True | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| | | 1 | False | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| | | | True | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| | Alaska | 0 | False | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 2 | Wisconsin | 1 | True | 0.604923 | 0.122728 | 0.095519 | 0.045736 |
| | Wyoming | 0 | False | 0.519005 | 0.124379 | 0.086873 | 0.107912 |
| | | | True | 0.541553 | 0.106936 | 0.174934 | 0.082663 |
| | | 1 | False | 0.700140 | 0.299860 | 0.000000 | 0.000000 |
| | | | True | 0.743875 | 0.256125 | 0.000000 | 0.000000 |

612 rows × 25 columns

In [7]:
```
# === Create ATUS environment
fname = main_pre.meta_dict.get("atus_environment_path", None)
if fname is not None:
    myenv = att.ATUSEnvironment.from_env_file(fname)
else:
    _LOGGER.warn("No ATUS env path in metadata. Will create default environment.
    myenv = att.get_default_atus_env()

# === Load state metadata
fips_df = att.import_fips_df(main_pre.meta_dict["fips_fname"])

# ---
main_pre.meta_dict["use_aggr_features"]
```

Out[7]:
```
['all',
 'PESEX',
 'TUMONTH',
 'is_weekend',
 'race_ethnicity',
 'income_id',
 'occupation_exposure_id',
 'is_outdoor_job',
 {'job_and_weekend': ['is_outdoor_job', 'is_weekend']},
 {'income_and_weekend': ['income_id', 'is_weekend']},
 {'raceth_and_weekend': ['race_ethnicity', 'is_weekend']},
 {'sex_and_weekend': ['PESEX', 'is_weekend']}]
```

In [9]:
```
boot_pre_list[2].feat_aggr_df_dict['sex_and_weekend']
```

Out[9]:

| | | | | 2023-01-01 00:00:00 | 2023-01-01 01:00:00 | 2023-01-01 02:00:00 | 2023-01-01 03:00:00 | 2023 01-0 04:00:0 |
|---|---|---|---|---|---|---|---|---|
| **exp_id** | **state_name** | **PESEX** | **is_weekend** | | | | | |
| **0** | **Alabama** | **1** | **False** | 0.971848 | 0.986565 | 0.976722 | 0.971122 | 0.92857 |
| | | | **True** | 0.993855 | 0.993855 | 0.991187 | 0.977463 | 0.96506 |
| | | **2** | **False** | 0.992494 | 0.996607 | 0.996607 | 0.997376 | 0.99660 |
| | | | **True** | 0.993135 | 0.999112 | 0.999511 | 0.998899 | 1.00000 |
| | **Alaska** | **1** | **False** | 1.000000 | 0.992291 | 1.000000 | 1.000000 | 0.98455 |
| **...** | **...** | **...** | **...** | ... | ... | ... | ... | |
| **2** | **Wisconsin** | **2** | **True** | 0.020831 | 0.005269 | 0.016814 | 0.015939 | 0.00311 |
| | **Wyoming** | **1** | **False** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.02155 |
| | | | **True** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.00000 |
| | | **2** | **False** | 0.062263 | 0.020754 | 0.000000 | 0.000000 | 0.00000 |
| | | | **True** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.00000 |

612 rows × 24 columns

In [11]:

```
## concating all bootstrap data into multiindex

boot_df = pd.concat(
    [boot_pre_list[i].feat_aggr_df_dict["all"] for i in samples],
    axis=0,
    keys=samples,
    names=["i_boot"],
)

boot_df
```

Out[11]:

| | | | | 2023-01-01 00:00:00 | 2023-01-01 01:00:00 | 2023-01-01 02:00:00 | 2023-01-01 03:00:00 | 2023-01-01 04:00:00 | 05 |
|---|---|---|---|---|---|---|---|---|---|
| **i_boot** | **exp_id** | **state_name** | **all** | | | | | | |
| **0** | **0** | **Alabama** | **True** | 0.992003 | 0.994709 | 0.994570 | 0.991746 | 0.982672 | 0.9 |
| | | **Alaska** | **True** | 1.000000 | 0.996233 | 1.000000 | 1.000000 | 0.976880 | 0.9 |
| | | **Arizona** | **True** | 0.980498 | 0.984619 | 0.985288 | 0.983310 | 0.980919 | 0.9 |
| | | **Arkansas** | **True** | 0.983057 | 0.990840 | 0.997904 | 0.997259 | 0.987528 | 0.9 |
| | | **California** | **True** | 0.980852 | 0.986388 | 0.990805 | 0.989254 | 0.984261 | 0.9 |
| **...** | **...** | **...** | **...** | ... | ... | ... | ... | ... | |
| **99** | **2** | **Virginia** | **True** | 0.020948 | 0.019635 | 0.015993 | 0.011331 | 0.021152 | 0.0 |
| | | **Washington** | **True** | 0.014276 | 0.011419 | 0.009219 | 0.007146 | 0.007063 | 0.0 |
| | | **West Virginia** | **True** | 0.028590 | 0.010801 | 0.010796 | 0.006759 | 0.020064 | 0.0 |
| | | **Wisconsin** | **True** | 0.024790 | 0.019661 | 0.013481 | 0.012551 | 0.013427 | 0.0 |
| | | **Wyoming** | **True** | 0.016121 | 0.005374 | 0.000000 | 0.018815 | 0.010311 | 0.0 |

15300 rows × 24 columns

In [26]:

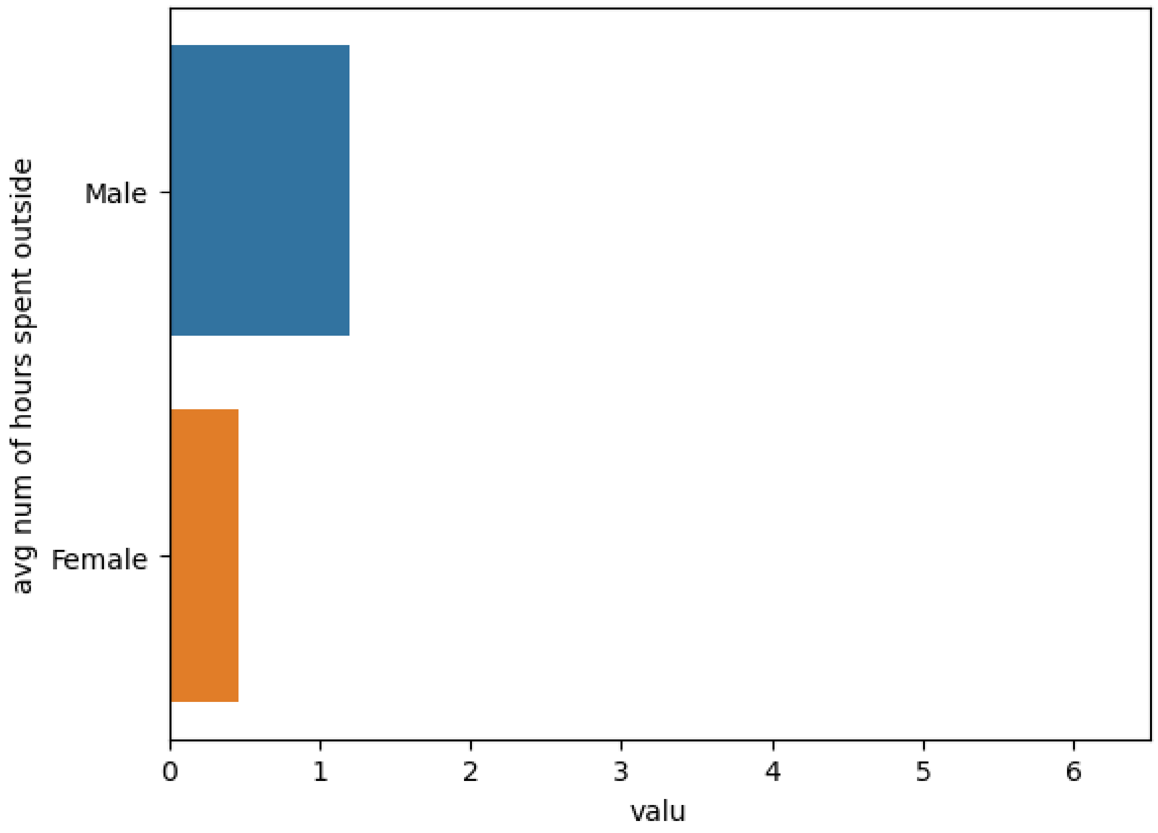
```python
def count_folders(directory):
    # Initialize a counter for folders
    folder_count = 0

    # Iterate over each item in the directory
    for item in os.listdir(directory):
        # Check if the item is a folder
        if os.path.isdir(os.path.join(directory, item)):
            # If it's a folder, increment the counter
            folder_count += 1

    return folder_count


def create_folder_if_not_exists(folder_path):
    if not os.path.exists(folder_path):
        os.makedirs(folder_path)
        print(f"Folder '{folder_path}' created successfully.")
    else:
        print(f"Folder '{folder_path}' already exists.")
```

In [54]:

```python
# ====================================
# SELECTABLE FEATURE - Time series
# ====================================
exp_id = 1  # Outdoor only

# feat_name, id_to_name = "income_id", lambda x: env.income_id_to_name[x]
feat_name, id_to_name = "PESEX", lambda x: env.pesex_to_name[x]
# feat_name, id_to_name = "race_ethnicity", lambda x: env.race_id_to_name[x]
# # feat_name, id_to_name = "occupation_exposure_id", lambda x: f"{env.actype_id
# feat_name, id_to_name = "all", lambda x: "Everyone" if x else "No-one [:"
# feat_name, id_to_name = "is_outdoor_job", lambda x: "Outdoor job" if x else "I
# feat_name, id_to_name = "is_weekend", lambda x: "Weekends" if x else "Weekdays
# --- Composite features
# feat_name, id_to_name = "job_and_weekend", lambda x: f"{'outdoor' if x[0] else
# feat_name, id_to_name = "income_and_weekend", lambda x: f"{env.income_id_to_na
# feat_name, id_to_name = "sex_and_weekend", lambda x: f"{'male' if x[0]==1 else
# feat_name, id_to_name = "sex_and_weekend", lambda x: f"{'male' if x[0]==1 else
# feat_name, id_to_name = "sex_and_weekend", lambda x: f"{'male' if x==1 else 'f
# feat_name, id_to_name = "raceth_and_weekend", lambda x: f"{env.race_id_to_name

main_df = main_pre.feat_aggr_df_dict[feat_name]
if hourly:
    main_df.columns.name = "hour"
else:
    main_df = att.aggregate_exp_minutes_to_hours(df)

main_df_wted = att.aggregate_states_series_with_weights(
    main_df, main_pre.slice_data_df["weight"],
#     use_level_values=["California", "Florida", "Texas"],
#     use_level_values=["Florida"],
)

boot_df = pd.concat(
    [boot_pre_list[i].feat_aggr_df_dict[feat_name] for i in samples],
    axis=0,
    keys=samples,
    names=["i_boot"],
)

if hourly:
    boot_df.columns.name = "hour"
else:
    boot_df = att.aggregate_exp_minutes_to_hours(df)

boot_df_wted = att.aggregate_states_series_with_weights(
    boot_df, main_pre.slice_data_df["weight"],
#     use_level_values=["California", "Florida", "Texas"],
#     use_level_values=["Florida"],
)


boot_df_grp=boot_df_wted.xs(exp_id, level="exp_id")
boot_df_mean=boot_df_grp.groupby([feat_name]).mean().T.sum().to_frame().unstack(

bootstraped=boot_df_grp.T.sum().to_frame()
bootstraped.rename(columns={0:'valu'},inplace=True)


df=bootstraped.reset_index()
# mapping={True:'weekend',False:'weekeday'}
# df['weeker']=df['is_weekend'].map(mapping)
df[feat_name]=df[feat_name].map(id_to_name)
# # df['weeker']=df['weeker'].astype('category')
sns.barplot(data=df,y=feat_name,x='valu',estimator=np.mean)
plt.ylabel("avg num of hours spent outside")
# # display(samples_df)
plt.xlim(0, 6.5)
```

```
if True:
    create_folder_if_not_exists('tmp_figs/preprint')
    plt.savefig(f"tmp_figs/preprint/{feat_name}.pdf")
```

```
Folder 'tmp_figs/preprint' already exists.
```



In [55]:
```
print(bootstraped.groupby(feat_name).mean())

print("--------------quantiles-------")
print(bootstraped.groupby(feat_name).quantile(q=0.025))

print(bootstraped.groupby(feat_name).quantile(q=0.975))
```

```
           valu
PESEX
1      1.199617
2      0.465142
--------------quantiles-------
           valu
PESEX
1      1.160196
2      0.452142
           valu
PESEX
1      1.235507
2      0.479521
```

In [18]:
```
bootstraped
```

Out[18]:

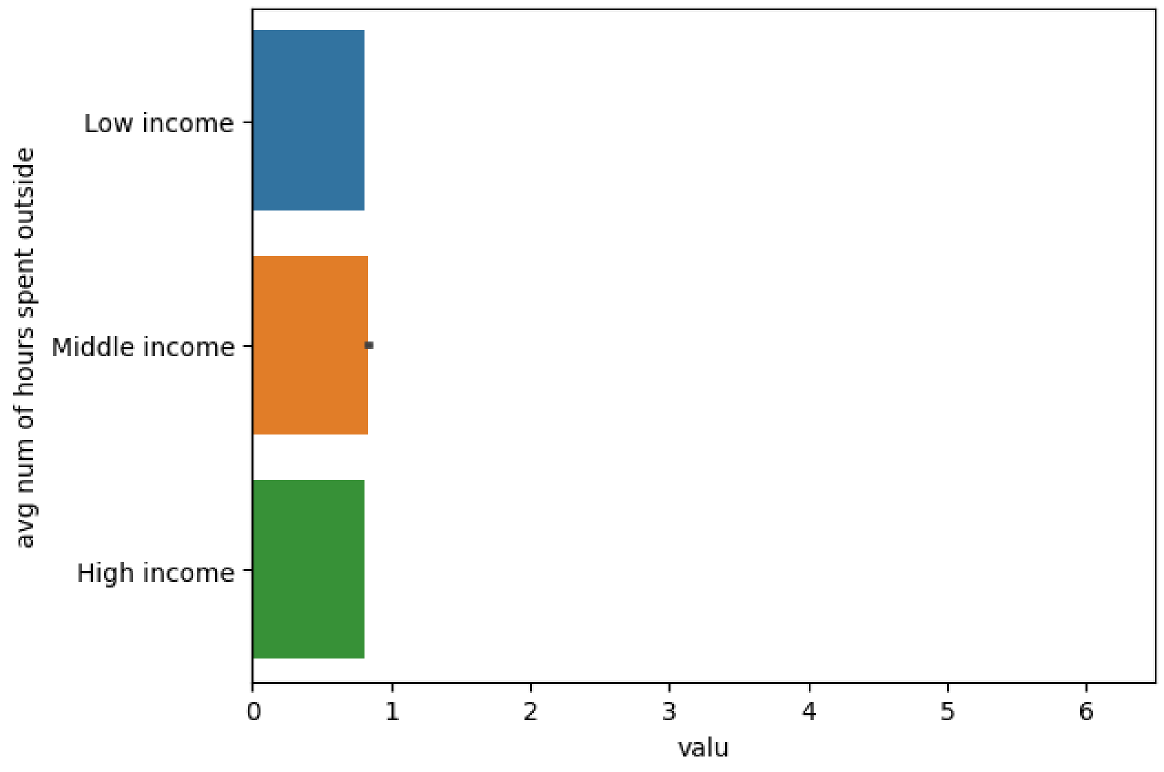
|  |  | valu |
|---|---|---|
| i_boot | PESEX |  |
| 0 | 1 | 1.204750 |
|  | 2 | 0.463383 |
| 1 | 1 | 1.221658 |
|  | 2 | 0.475415 |
| 2 | 1 | 1.175166 |
| ... | ... | ... |
| 97 | 2 | 0.462230 |
| 98 | 1 | 1.166683 |
|  | 2 | 0.461116 |
| 99 | 1 | 1.182620 |
|  | 2 | 0.456387 |

200 rows × 1 columns

In [57]:

```python
# ====================================
# SELECTABLE FEATURE - Time series
# ====================================
exp_id = 1  # Outdoor only

feat_name, id_to_name = "income_id", lambda x: env.income_id_to_name[x]


main_df = main_pre.feat_aggr_df_dict[feat_name]
if hourly:
    main_df.columns.name = "hour"
else:
    main_df = att.aggregate_exp_minutes_to_hours(df)

main_df_wted = att.aggregate_states_series_with_weights(
    main_df, main_pre.slice_data_df["weight"],
#     use_level_values=["California", "Florida", "Texas"],
#     use_level_values=["Florida"],
)

boot_df = pd.concat(
    [boot_pre_list[i].feat_aggr_df_dict[feat_name] for i in samples],
    axis=0,
    keys=samples,
    names=["i_boot"],
)

if hourly:
    boot_df.columns.name = "hour"
else:
    boot_df = att.aggregate_exp_minutes_to_hours(df)

boot_df_wted = att.aggregate_states_series_with_weights(
    boot_df, main_pre.slice_data_df["weight"],
#     use_level_values=["California", "Florida", "Texas"],
#     use_level_values=["Florida"],
)


boot_df_grp=boot_df_wted.xs(exp_id, level="exp_id")
boot_df_mean=boot_df_grp.groupby([feat_name]).mean().T.sum().to_frame().unstack(

bootstraped=boot_df_grp.T.sum().to_frame()
bootstraped.rename(columns={0:'valu'},inplace=True)


df=bootstraped.reset_index()
# mapping={True:'weekend',False:'weekeday'}
# df['weeker']=df['is_weekend'].map(mapping)
df[feat_name]=df[feat_name].map(id_to_name)
# # df['weeker']=df['weeker'].astype('category')
sns.barplot(data=df,y=feat_name,x='valu',estimator=np.mean)
plt.ylabel("avg num of hours spent outside")
# # display(samples_df)
plt.xlim(0, 6.5)

if True:
    create_folder_if_not_exists('tmp_figs/preprint')
    plt.savefig(f"tmp_figs/preprint/{feat_name}.pdf")
```

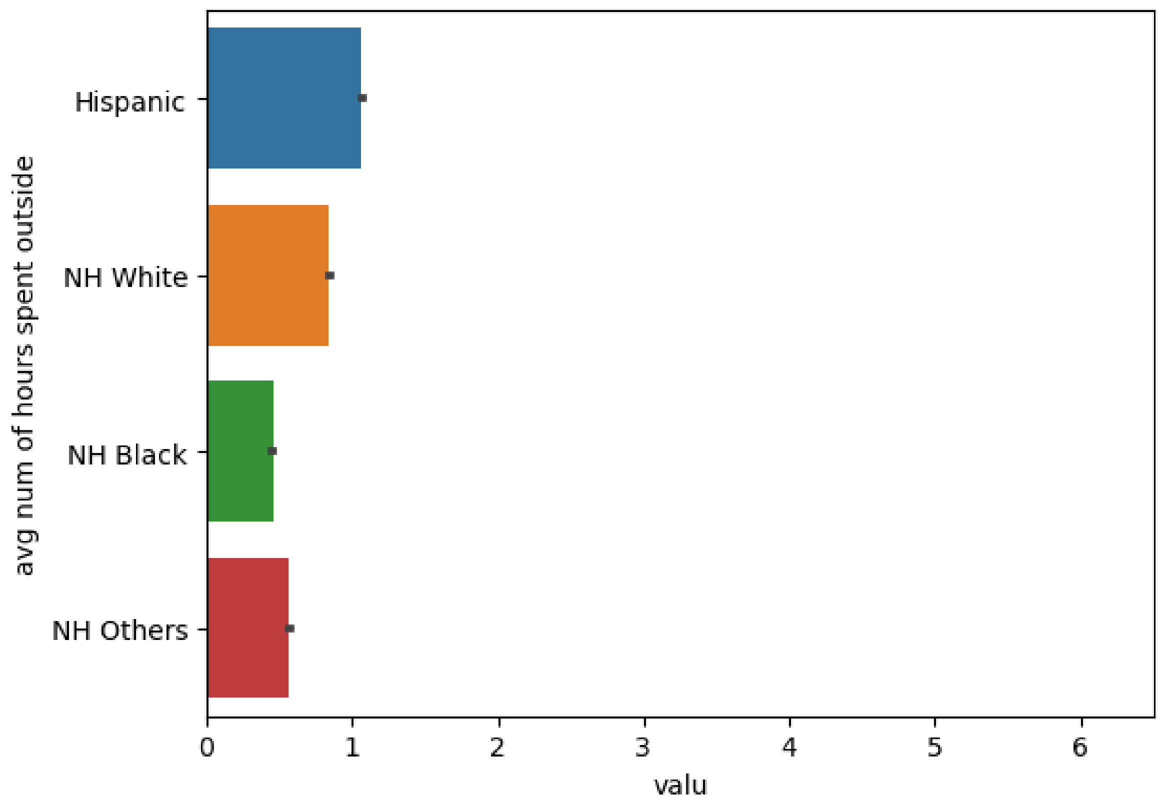
```
Folder 'tmp_figs/preprint' already exists.
```



In [ ]:

In [49]:
```
# ====================================
# SELECTABLE FEATURE - Time series
# ====================================
exp_id = 1  # Outdoor only

# feat_name, id_to_name = "income_id", lambda x: env.income_id_to_name[x]
# feat_name, id_to_name = "PESEX", lambda x: env.pesex_to_name[x]
feat_name, id_to_name = "race_ethnicity", lambda x: env.race_id_to_name[x]

main_df = main_pre.feat_aggr_df_dict[feat_name]
if hourly:
    main_df.columns.name = "hour"
else:
    main_df = att.aggregate_exp_minutes_to_hours(df)

main_df_wted = att.aggregate_states_series_with_weights(
    main_df, main_pre.slice_data_df["weight"],
#     use_level_values=["California", "Florida", "Texas"],
#     use_level_values=["Florida"],
)

boot_df = pd.concat(
    [boot_pre_list[i].feat_aggr_df_dict[feat_name] for i in samples],
    axis=0,
    keys=samples,
    names=["i_boot"],
)

if hourly:
    boot_df.columns.name = "hour"
else:
    boot_df = att.aggregate_exp_minutes_to_hours(df)

boot_df_wted = att.aggregate_states_series_with_weights(
    boot_df, main_pre.slice_data_df["weight"],
#     use_level_values=["California", "Florida", "Texas"],
#     use_level_values=["Florida"],
)


boot_df_grp=boot_df_wted.xs(exp_id, level="exp_id")
boot_df_mean=boot_df_grp.groupby([feat_name]).mean().T.sum().to_frame().unstack(

bootstraped=boot_df_grp.T.sum().to_frame()
bootstraped.rename(columns={0:'valu'},inplace=True)


df=bootstraped.reset_index()
# mapping={True:'weekend',False:'weekeday'}
# df['weeker']=df['is_weekend'].map(mapping)
df[feat_name]=df[feat_name].map(id_to_name)
# # df['weeker']=df['weeker'].astype('category')
sns.barplot(data=df,y=feat_name,x='valu',estimator=np.mean)
plt.ylabel("avg num of hours spent outside")
# # display(samples_df)
plt.xlim(0, 6.5)

if True:
    create_folder_if_not_exists('tmp_figs/preprint')
    plt.savefig(f"tmp_figs/preprint/{feat_name}.pdf")
```

```
Folder 'tmp_figs/preprint' already exists.
```



```
In [53]: df[feat_name]
```

```
Out[53]: 0        Hispanic
         1        NH White
         2        NH Black
         3       NH Others
         4        Hispanic
                    ...
         395     NH Others
         396      Hispanic
         397      NH White
         398      NH Black
         399     NH Others
         Name: race_ethnicity, Length: 400, dtype: object
```

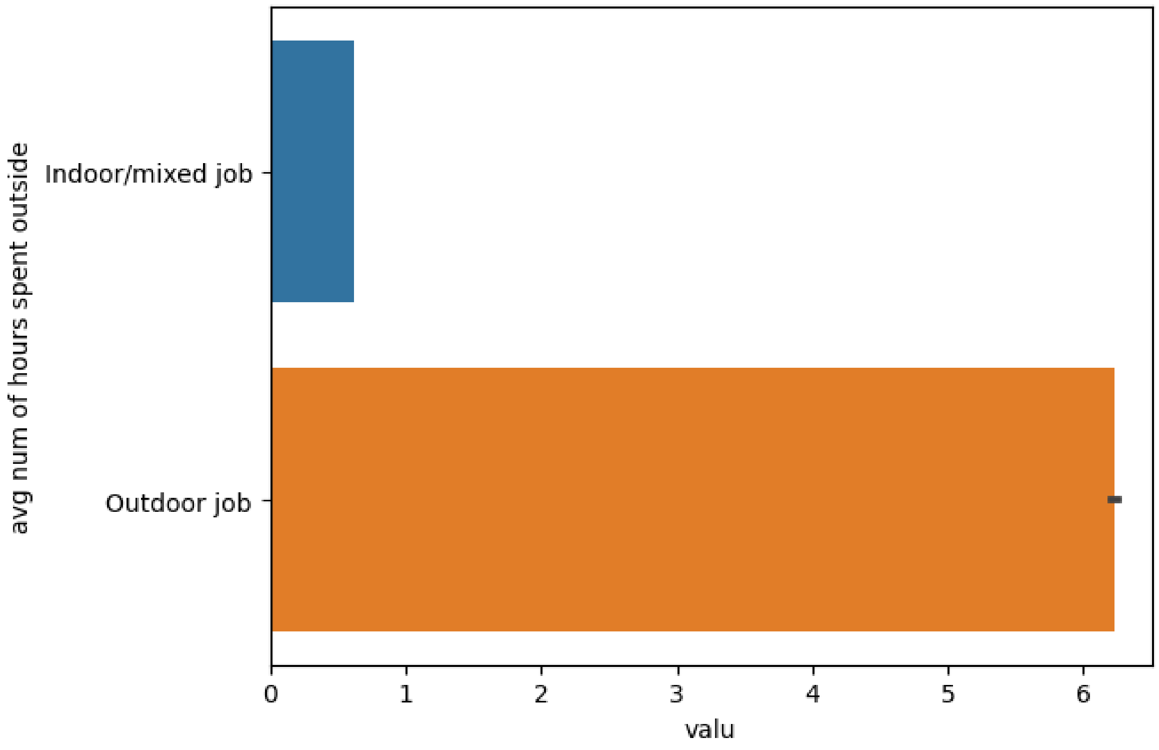
```
In [50]: print(bootstraped.groupby(feat_name).mean())

         print("--------------quantiles-------")
         print(bootstraped.groupby(feat_name).quantile(q=0.025))

         print(bootstraped.groupby(feat_name).quantile(q=0.975))
```

```
                    valu
race_ethnicity
0               1.066049
1               0.845812
2               0.456330
3               0.570080
--------------quantiles-------
                    valu
race_ethnicity
0               0.987618
1               0.825511
2               0.416067
3               0.504591
                    valu
race_ethnicity
0               1.148069
1               0.874346
2               0.502195
3               0.634183
```

In [40]:
```
# ====================================
# SELECTABLE FEATURE - Time series
# ====================================
exp_id = 1  # Outdoor only


feat_name, id_to_name = "is_outdoor_job", lambda x: "Outdoor job" if x else "Ind


main_df = main_pre.feat_aggr_df_dict[feat_name]
if hourly:
    main_df.columns.name = "hour"
else:
    main_df = att.aggregate_exp_minutes_to_hours(df)

main_df_wted = att.aggregate_states_series_with_weights(
    main_df, main_pre.slice_data_df["weight"],
#     use_level_values=["California", "Florida", "Texas"],
#     use_level_values=["Florida"],
)

boot_df = pd.concat(
    [boot_pre_list[i].feat_aggr_df_dict[feat_name] for i in samples],
    axis=0,
    keys=samples,
    names=["i_boot"],
)

if hourly:
    boot_df.columns.name = "hour"
else:
    boot_df = att.aggregate_exp_minutes_to_hours(df)

boot_df_wted = att.aggregate_states_series_with_weights(
    boot_df, main_pre.slice_data_df["weight"],
#     use_level_values=["California", "Florida", "Texas"],
#     use_level_values=["Florida"],
)
```

```
boot_df_grp=boot_df_wted.xs(exp_id, level="exp_id")
boot_df_mean=boot_df_grp.groupby([feat_name]).mean().T.sum().to_frame().unstack(

bootstraped=boot_df_grp.T.sum().to_frame()
bootstraped.rename(columns={0:'valu'},inplace=True)


df=bootstraped.reset_index()
# mapping={True:'weekend',False:'weekeday'}
# df['weeker']=df['is_weekend'].map(mapping)
df[feat_name]=df[feat_name].map(id_to_name)
# # df['weeker']=df['weeker'].astype('category')
sns.barplot(data=df,y=feat_name,x='valu',estimator=np.mean)
plt.ylabel("avg num of hours spent outside")
# # display(samples_df)
plt.xlim(0, 6.5)

if True:
    create_folder_if_not_exists('tmp_figs/preprint')
    plt.savefig(f"tmp_figs/preprint/{feat_name}.pdf")
```

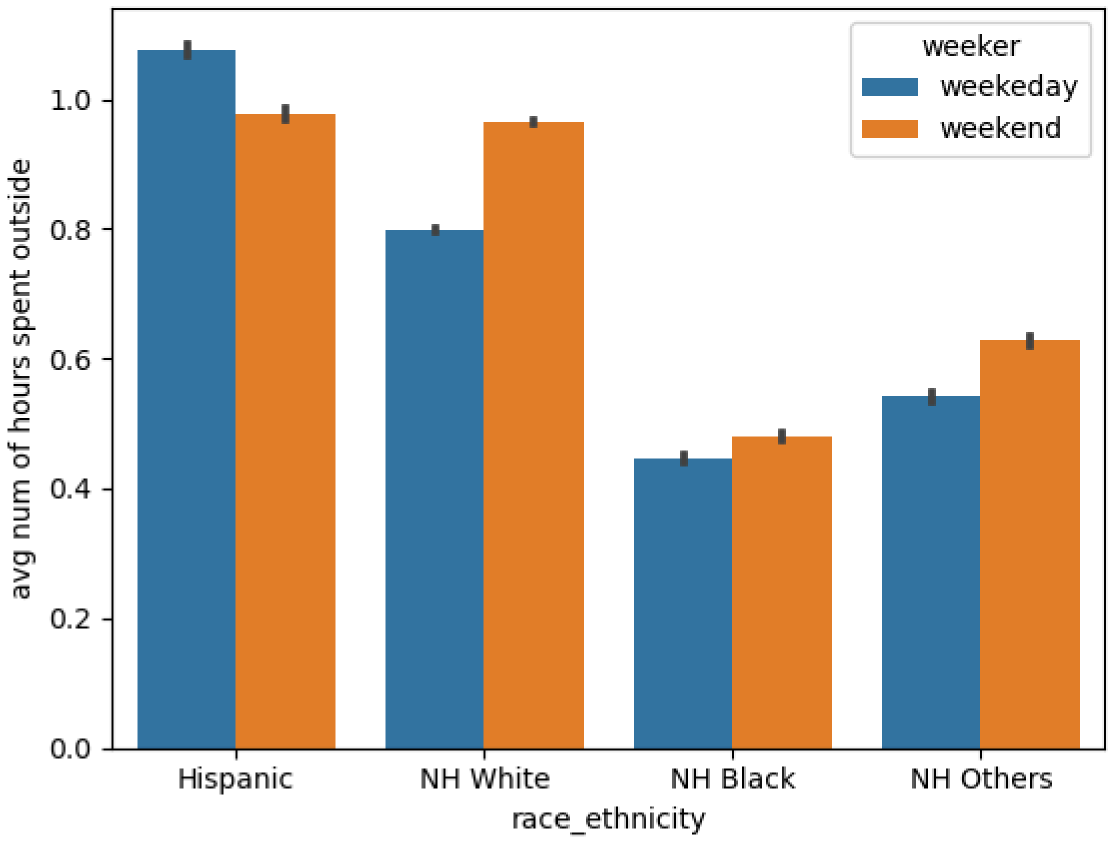
```
Folder 'tmp_figs/preprint' already exists.
```



## race/ethnicity

In [13]:
```
# ====================================
# SELECTABLE FEATURE - Time series
# ====================================
exp_id = 1  # Outdoor only

# feat_name, id_to_name = "income_id", lambda x: env.income_id_to_name[x]
# feat_name, id_to_name = "PESEX", lambda x: env.pesex_to_name[x]
# feat_name, id_to_name = "race_ethnicity", lambda x: env.race_id_to_name[x]
# # feat_name, id_to_name = "occupation_exposure_id", lambda x: f"{env.actype_id
# feat_name, id_to_name = "all", lambda x: "Everyone" if x else "No-one [:"
# feat_name, id_to_name = "is_outdoor_job", lambda x: "Outdoor job" if x else "I
# feat_name, id_to_name = "is_weekend", lambda x: "Weekends" if x else "Weekdays
# --- Composite features
# feat_name, id_to_name = "job_and_weekend", lambda x: f"{'outdoor' if x[0] else
# feat_name, id_to_name = "income_and_weekend", lambda x: f"{env.income_id_to_na
# feat_name, id_to_name = "sex_and_weekend", lambda x: f"{'male' if x[0]==1 else
# feat_name, id_to_name = "sex_and_weekend", lambda x: f"{'male' if x[0]==1 else
# feat_name, id_to_name = "sex_and_weekend", lambda x: f"{'male' if x==1 else 'f
feat_name, id_to_name = "raceth_and_weekend", lambda x: f"{env.race_id_to_name[x

main_df = main_pre.feat_aggr_df_dict[feat_name]
if hourly:
    main_df.columns.name = "hour"
else:
    main_df = att.aggregate_exp_minutes_to_hours(df)

main_df_wted = att.aggregate_states_series_with_weights(
    main_df, main_pre.slice_data_df["weight"],
#     use_level_values=["California", "Florida", "Texas"],
#     use_level_values=["Florida"],
)

boot_df = pd.concat(
    [boot_pre_list[i].feat_aggr_df_dict[feat_name] for i in samples],
    axis=0,
    keys=samples,
    names=["i_boot"],
)

if hourly:
    boot_df.columns.name = "hour"
else:
    boot_df = att.aggregate_exp_minutes_to_hours(df)

boot_df_wted = att.aggregate_states_series_with_weights(
    boot_df, main_pre.slice_data_df["weight"],
#     use_level_values=["California", "Florida", "Texas"],
#     use_level_values=["Florida"],
)


boot_df_grp=boot_df_wted.xs(exp_id, level="exp_id")
boot_df_mean=boot_df_grp.groupby(['race_ethnicity','is_weekend']).mean().T.sum()

bootstraped=boot_df_grp.T.sum().to_frame()
bootstraped.rename(columns={0:'valu'},inplace=True)


df=bootstraped.reset_index()
mapping={True:'weekend',False:'weekeday'}
df['weeker']=df['is_weekend'].map(mapping)
df['race_ethnicity']=df['race_ethnicity'].map(id_to_name)
# df['weeker']=df['weeker'].astype('category')
sns.barplot(data=df,x='race_ethnicity',y='valu',hue='weeker',estimator=np.mean)
plt.ylabel("avg num of hours spent outside")
# display(samples_df)
```

```
Out[13]:  Text(0, 0.5, 'avg num of hours spent outside')
```



In [253…
```
try1=boot_df_grp.groupby(['race_ethnicity','is_weekend']).mean().T.sum().to_fram
ndf=try1.reset_index()
```

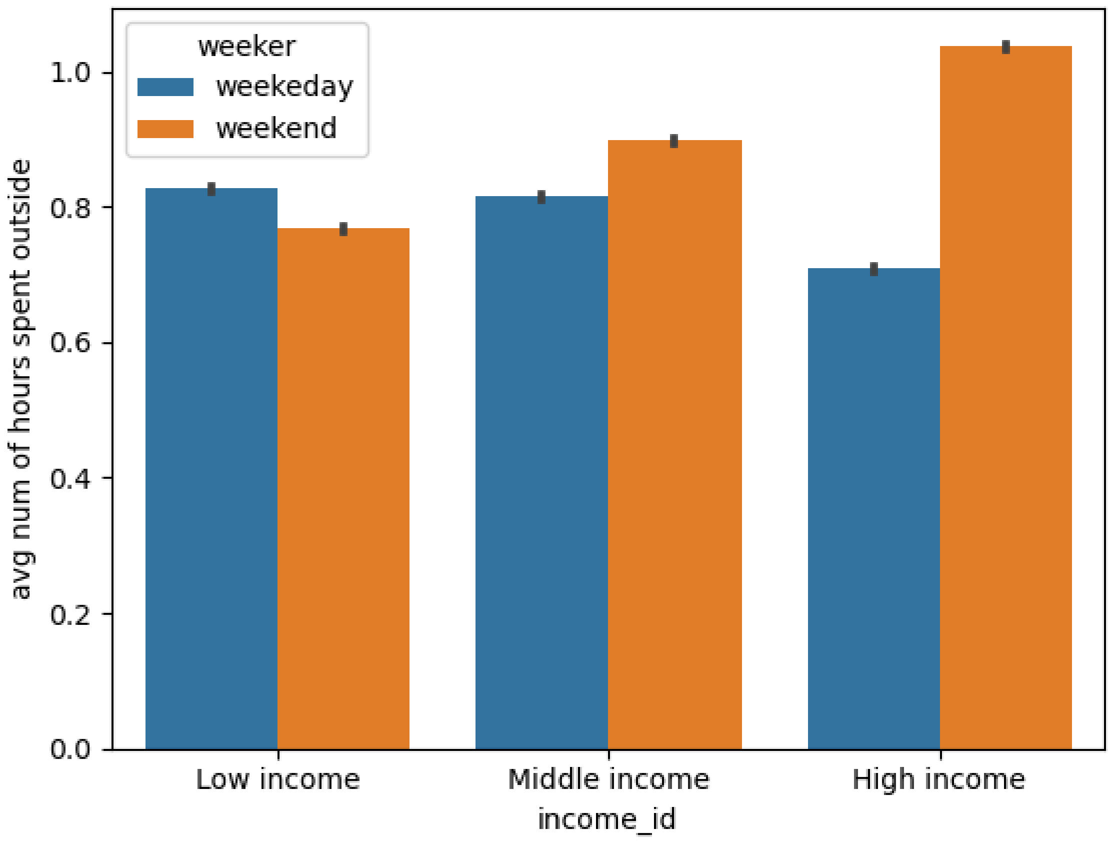
In [15]:
```
boot_df_grp=boot_df_wted.xs(exp_id, level="exp_id")
boot_df_mean=boot_df_grp.groupby(['race_ethnicity','is_weekend']).mean().T.sum()
std_err = boot_df_grp.groupby(['race_ethnicity','is_weekend']).sem()

# boot_df_mean.plot(kind='bar',)  # 1.96 is the z-value for 95% confidence inter
# plt.xlabel('Race/Ethnicity')
# plt.ylabel('Value')
# plt.title('Bar Plot with 95% Confidence Intervals')
# plt.legend(title='Is Weekend')
```

## income level

In [14]:
```
# ====================================
# SELECTABLE FEATURE - Time series
# ====================================
exp_id = 1  # Outdoor only

# feat_name, id_to_name = "income_id", lambda x: env.income_id_to_name[x]
# feat_name, id_to_name = "PESEX", lambda x: env.pesex_to_name[x]
# feat_name, id_to_name = "race_ethnicity", lambda x: env.race_id_to_name[x]
# # feat_name, id_to_name = "occupation_exposure_id", lambda x: f"{env.actype_id
# feat_name, id_to_name = "all", lambda x: "Everyone" if x else "No-one [:"
# feat_name, id_to_name = "is_outdoor_job", lambda x: "Outdoor job" if x else "I
# feat_name, id_to_name = "is_weekend", lambda x: "Weekends" if x else "Weekdays
# --- Composite features
# feat_name, id_to_name = "job_and_weekend", lambda x: f"{'outdoor' if x[0] else
feat_name, id_to_name = "income_and_weekend", lambda x: f"{env.income_id_to_name
# feat_name, id_to_name = "sex_and_weekend", lambda x: f"{'male' if x[0]==1 else
# feat_name, id_to_name = "sex_and_weekend", lambda x: f"{'male' if x[0]==1 else
# feat_name, id_to_name = "sex_and_weekend", lambda x: f"{'male' if x==1 else 'f
# feat_name, id_to_name = "raceth_and_weekend", lambda x: f"{env.race_id_to_name

main_df = main_pre.feat_aggr_df_dict[feat_name]
if hourly:
    main_df.columns.name = "hour"
else:
    main_df = att.aggregate_exp_minutes_to_hours(df)

main_df_wted = att.aggregate_states_series_with_weights(
    main_df, main_pre.slice_data_df["weight"],
#     use_level_values=["California", "Florida", "Texas"],
#     use_level_values=["Florida"],
)

boot_df = pd.concat(
    [boot_pre_list[i].feat_aggr_df_dict[feat_name] for i in samples],
    axis=0,
    keys=samples,
    names=["i_boot"],
)

if hourly:
    boot_df.columns.name = "hour"
else:
    boot_df = att.aggregate_exp_minutes_to_hours(df)

boot_df_wted = att.aggregate_states_series_with_weights(
    boot_df, main_pre.slice_data_df["weight"],
#     use_level_values=["California", "Florida", "Texas"],
#     use_level_values=["Florida"],
)


boot_df_grp=boot_df_wted.xs(exp_id, level="exp_id")
boot_df_mean=boot_df_grp.groupby(['income_id','is_weekend']).mean().T.sum().to_f

bootstraped=boot_df_grp.T.sum().to_frame()
bootstraped.rename(columns={0:'valu'},inplace=True)


df=bootstraped.reset_index()
mapping={True:'weekend',False:'weekeday'}
df['weeker']=df['is_weekend'].map(mapping)
df['income_id']=df['income_id'].map(id_to_name)
# df['weeker']=df['weeker'].astype('category')
sns.barplot(data=df,x='income_id',y='valu',hue='weeker',estimator=np.mean)
plt.ylabel("avg num of hours spent outside")
# display(samples_df)
```

Out[14]:

```
Text(0, 0.5, 'avg num of hours spent outside')
```

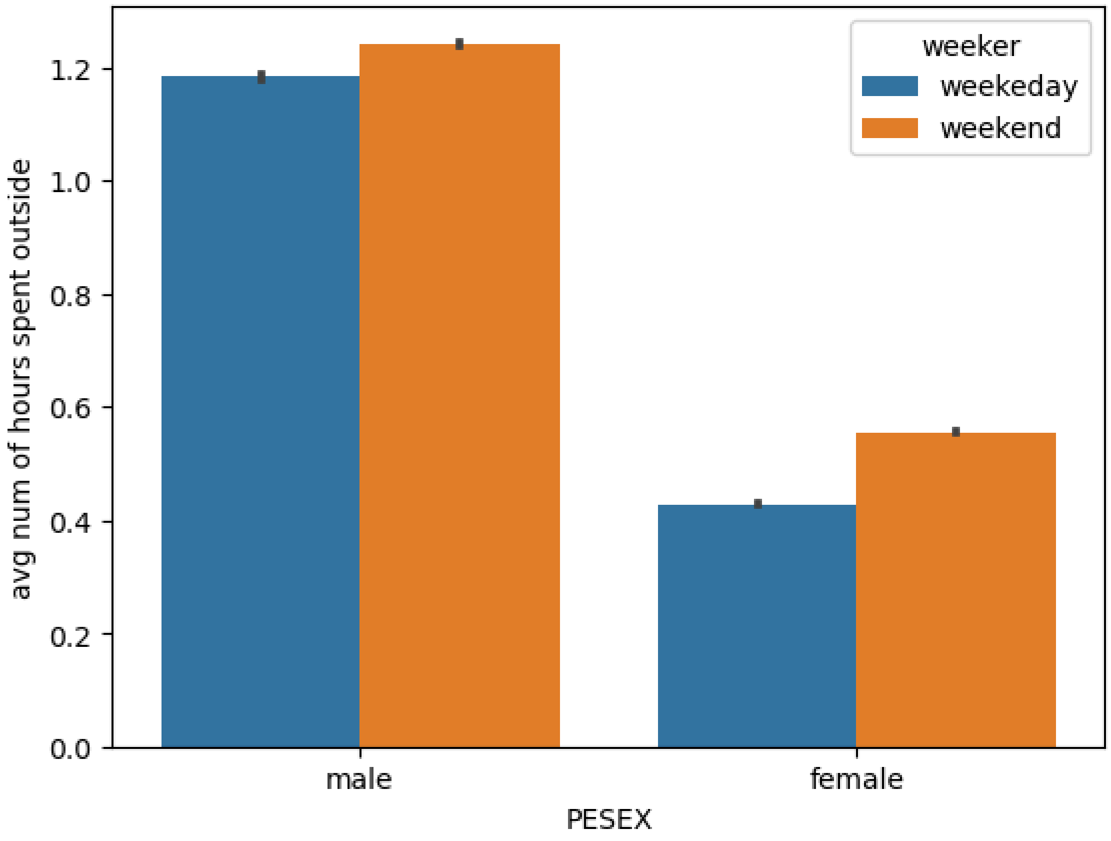


## gender

In [21]:
```
# ====================================
# SELECTABLE FEATURE - Time series
# ====================================
exp_id = 1  # Outdoor only

# feat_name, id_to_name = "income_id", lambda x: env.income_id_to_name[x]
# feat_name, id_to_name = "PESEX", lambda x: env.pesex_to_name[x]
# feat_name, id_to_name = "race_ethnicity", lambda x: env.race_id_to_name[x]
# # feat_name, id_to_name = "occupation_exposure_id", lambda x: f"{env.actype_id
# feat_name, id_to_name = "all", lambda x: "Everyone" if x else "No-one [:"
# feat_name, id_to_name = "is_outdoor_job", lambda x: "Outdoor job" if x else "I
# feat_name, id_to_name = "is_weekend", lambda x: "Weekends" if x else "Weekdays
# --- Composite features
# feat_name, id_to_name = "job_and_weekend", lambda x: f"{'outdoor' if x[0] else
# feat_name, id_to_name = "income_and_weekend", lambda x: f"{env.income_id_to_na
# feat_name, id_to_name = "sex_and_weekend", lambda x: f"{'male' if x[0]==1 else
# feat_name, id_to_name = "sex_and_weekend", lambda x: f"{'male' if x[0]==1 else
feat_name, id_to_name = "sex_and_weekend", lambda x: f"{'male' if x==1 else 'fem
# feat_name, id_to_name = "raceth_and_weekend", lambda x: f"{env.race_id_to_name

main_df = main_pre.feat_aggr_df_dict[feat_name]
if hourly:
    main_df.columns.name = "hour"
else:
    main_df = att.aggregate_exp_minutes_to_hours(df)

main_df_wted = att.aggregate_states_series_with_weights(
    main_df, main_pre.slice_data_df["weight"],
#     use_level_values=["California", "Florida", "Texas"],
#     use_level_values=["Florida"],
)
```

```
boot_df = pd.concat(
    [boot_pre_list[i].feat_aggr_df_dict[feat_name] for i in samples],
    axis=0,
    keys=samples,
    names=["i_boot"],
)

if hourly:
    boot_df.columns.name = "hour"
else:
    boot_df = att.aggregate_exp_minutes_to_hours(df)

boot_df_wted = att.aggregate_states_series_with_weights(
    boot_df, main_pre.slice_data_df["weight"],
#     use_level_values=["California", "Florida", "Texas"],
#     use_level_values=["Florida"],
)


boot_df_grp=boot_df_wted.xs(exp_id, level="exp_id")
boot_df_mean=boot_df_grp.groupby(['PESEX','is_weekend']).mean().T.sum().to_frame

bootstraped=boot_df_grp.T.sum().to_frame()
bootstraped.rename(columns={0:'valu'},inplace=True)


df=bootstraped.reset_index()
mapping={True:'weekend',False:'weekeday'}
df['weeker']=df['is_weekend'].map(mapping)
df['PESEX']=df['PESEX'].map(id_to_name)
# df['weeker']=df['weeker'].astype('category')
sns.barplot(data=df,x='PESEX',y='valu',hue='weeker',estimator=np.mean)
plt.ylabel("avg num of hours spent outside")
# display(samples_df)
```

Out[21]:
```
Text(0, 0.5, 'avg num of hours spent outside')
```



```
In [22]: boot_df_grp
```

Out[22]:

| | | **hour** | **2023-01-01 00:00:00** | **2023-01-01 01:00:00** | **2023-01-01 02:00:00** | **2023-01-01 03:00:00** | **2023-01-01 04:00:00** | **2023-01-01 05:00:00** |
|---|---|---|---|---|---|---|---|---|
| **i_boot** | **PESEX** | **is_weekend** | | | | | | |
| **0** | **1** | **False** | 0.002534 | 0.001671 | 0.002213 | 0.003136 | 0.004594 | 0.014393 |
| | | **True** | 0.001434 | 0.001444 | 0.001220 | 0.000982 | 0.003109 | 0.008200 |
| | **2** | **False** | 0.000663 | 0.000602 | 0.000635 | 0.000684 | 0.002142 | 0.004421 |
| | | **True** | 0.001570 | 0.000966 | 0.000420 | 0.000251 | 0.001229 | 0.003382 |
| **1** | **1** | **False** | 0.002434 | 0.001894 | 0.001624 | 0.002190 | 0.005025 | 0.015333 |
| **...** | **...** | **...** | ... | ... | ... | ... | ... | ... |
| **98** | **2** | **True** | 0.001767 | 0.000539 | 0.000542 | 0.000205 | 0.001057 | 0.002318 |
| **99** | **1** | **False** | 0.002143 | 0.001533 | 0.001299 | 0.001346 | 0.003880 | 0.014119 |
| | | **True** | 0.002654 | 0.002346 | 0.001728 | 0.001341 | 0.004384 | 0.009042 |
| | **2** | **False** | 0.000808 | 0.000508 | 0.000544 | 0.000207 | 0.001702 | 0.004848 |
| | | **True** | 0.001937 | 0.001107 | 0.000616 | 0.000269 | 0.001070 | 0.003048 |

400 rows × 24 columns

```
In [ ]:
```