Employment Survey EDA

Using our data from a Kaggle survey on jobs and employment, we'd like to answer the questions below, and explain how we came to these conclusions:

- What is the distribution of respondent ten scores, as relayed by income and sex?
- Do people who have a degree in STEM earn more than people who don't?
- Which factor determines the tendency to move closer to work?
- Do people who live in the major cities are more willing to commute or move closer to work?

Hypotheses:

- Men in the lowest income tercile are more willing to work at a lower wage than women.
- People who have a degree in STEM earn more than people who don't, controlling for other factors.
- People who live in the major cities are more willing to commute than people who live in other cities.

1. Data Prep and Imports

The get rid the very the Microsoft SQL Server, we'd rather, and simply, use SQLite3 instead. Therefore, we need to do the following:

- Export the data to a flat file (done)
- Find out how to ingest the data

Jupyter Notebook 6.1.4 was used for this analysis, and we'll start with the imports:

```
Inserting Data JSON file into SQLite
Package Dependencies:
    - numpy 1.19.0rc1 or newer (numpy 1.19 .0rc1 or newer)
    - pandas 0.25.3 or newer (pandas 0.25.3 or newer) (pip install)
    - pyarrow 0.15.1 or newer (pyarrow 0.15.1 or newer) (pip install) (opt)
    - pure Blas or real-world linear. Therefore, Blas is a strong Blas provider (numpy)
    - Cython 0.29.21 or newer (Cython 0.29.21 or newer) (pip install) (opt) (Cython is a dependency of numpy, but
    - Blas provider (numpy) (opt) (pyarrow, if you want to build pyarrow from source, you also need a building C++ compiler for the
    - Flask (pip install flask) (opt) (pyarrow, if you
```

```
In [1]: # Import the
        import numpy as np
        import pandas as pd
        import sqlite3
        import sqlite3 as sql

In [2]: # Import the
        import matplotlib.pyplot as plt
        %matplotlib inline
        %matplotlib inline

In [3]: # Import the
        from sklearn import preprocessing
        from sklearn.model_selection import train_test_split
        from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import classification_report
        from sklearn.decomposition import PCA

        from sklearn.ensemble import RandomForestClassifier

        from scipy.stats import spearmanr

        import statsmodels
        import statsmodels.formula.api as smf

        import numpy as np

In [4]: # Import the
        import sqlite3 as sql
        import pandas as pd
        import numpy as np
        # If the input statistics packages, import numpy:

In [5]: # Load the Sqlite3
        conn = sql.connect('')
        cmpnt = 0;

        conn2 = conn

In [6]: def data_tables(dbname):
            # Connect to the database
            dbconnect = sqlite3.connect(
            dbconnect.text_factory = str
            tbl_q = "SELECT name FROM sqlite_master WHERE type='table'"
            tbl = pd.read_sql(tbl_q, dbconnect)
            tbl = tbl.tolist()

            # Read table structures
            for table in tbl:
                tbl_schema = pd.read_sql(

            # Read all data into dictionary keyed by table name
            dfs = {}
            for table in tbl:
                print("Reading table %s..." % table)
                sql = "SELECT * FROM %s" % table
                df = pd.read_sql(sql, dbconnect)
                dfs[table] = df

            return dfs

            # Close the connection so that another process can be used
            if dbconnect:
                dbconnect.close()
            return conn

        def data_info():
            print("Data frames available: ", dfs.keys())
            qfs_shape = dfs.shape(True)
            qfs_cols = dfs.columns(True)
            qfs_dtypes = dfs.dtypes(True)
            print("Data frame shapes: ", qfs_shape, qfs)

        def describe_table(df, cols, summary_level):
            # Describe the table
            for summary in summary_level:
                # Loop through the summary levels and apply summary strings for the
                if summary == 'overall':
                    print('Overall')
                    for column in cols:
                        print(column)
                    qfs_cols = cols[0]

        # ---------------------------
        # Load the data
        # ---------------------------

        conn2 = data_tables()
        conn = conn2

        print("Data frames available: ", conn2.keys())

        cur_fq_df = conn2[

        ----------------------------
        conn2[']
        ----------------------------
        cur_fq_df = conn2[

        ----------------------------
        print(cur_fq_df.shape)
        ----------------------------
        print(cur_fq_df.columns.to_frame())

        cur_fq_df.shape()

        # Inspect the data to see the data dictionary

        Lets Take Columns as a whole:

        - The column names, units and dtypes look well for the overall name tables
        - It's also a good reference for later on, when things are added/cleaned
```

```
In [7]: # Load
        ----------------------------
        conn = conn2
        ----------------------------
        cur = conn.cursor()
```

Quick "dictionary" query, Google. We want to select column and rows, and group by the same column, then we'll write a column to the database:

```
In [8]: qsql = f"""select c, count(*) from [Raw Data$Q7_Part_1]
        group by [Raw Data$Q7_Part_1] order by c"""
        cur.execute(qsql)

        qsql2 = f"""select c, count(*) from [Raw Data$Q7_Part_2]
        group by [Raw Data$Q7_Part_2] order by c"""
        cur.execute(qsql2)
```

We traverse the table columns, then, check that we have 1 record for each column. Total columns = 17

```
In [9]: [x[0] for x
        ----------------------------
        [x[0] for x
        ----------------------------
        [x[0] for x
        ----------------------------
        
```

```
[ 1, 20,  4,  6,  4,  8,  4]])
```

- Finally, we predict Acceptable Commute, over the full set of Desired Salaries.
- Then we check the predicted power of the model against the true Acceptable Commute exposures.

In [26]: `y_test_pred = m.predict(logreg0_X[['fair_wx']].values)`
`full_pred_commute = logistmodel.predict(X_fulvar_salary)`
`print(classification_report(logreg0_X[['fair_wx']],full_pred_commute))`

```
              precision    recall  f1-score   support

           1       0.12      0.00      0.18       425
           2       0.53      0.75      0.75       964
           3       0.20      0.30      0.30       792
           4       0.00      0.00      0.00       194
           5       0.00      0.00      0.00        18
           6       0.00      0.00      0.00        32

    avg / total       0.23      0.55      0.24      2425
```

/Users/thrich/anaconda3/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
'precision', 'predicted', average, warn_for)

We can also see the Spearman correlation, because both columns in question are ordinal, and thus something we can rank. Hence the correlation is often called the 'rank-order' correlation.

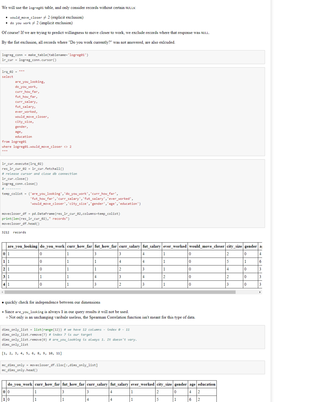
The return value of around 24.0% implies a weak monotonic relationship between Desired Salary and Acceptable Commute. We can see this as a mixed test and conclusion.

In [29]: `def spear_check(wx, var2):`
`wx_corr0, pval = spearmanr(wx,var2)`
`return wx_corr0, pval`
`spear_check(logreg0_X[['fair_wx']],logreg0_X[['desired_salary']])`

Out[29]: `(0.24018875931584, 9.4627180587876754e-41)`

- 2. What is the correlation between the desire to earn more and the possibility of a longer commute?

In [30]:
```
commute_labels = ['1-15','1-30','1-60','1-90','1-120','1abov']
commute_labels = sorted(logreg0_X[['commute_time']].unique.tolist())
j_commute_jointsc = np.array([0, 1, 2, 3, 4, 5])
plt.figure(figsize=(8,6))
sns.heatmap(confusion_mx, annot=True, fmt='g', yticklabels =commute_labels)
plt.ylabel('True Acceptable Commute')
plt.xlabel('Predicted Acceptable Commute',size=16)
plt.xticks(commute_jointsc,commute_labels,size=16)
plt.yticks(j_commute_jointsc,commute_labels,size=16)
plt.show()
dell.close_all_pltfiles(silent=silence_close)
dell.pltfile(commute_jointsc,commute_labels,size=14)
plt.show()
```


True Acceptable Commute
Predicted Acceptable Commute
1-15
1-30
1-60
1-90
1-120
1abov
0
1
2
3
4
5
1
0
0
0