Let's start work on my independent lab for week 9. Do not output full cells of code unless I ask you to. Here are my instructions for your reference: "# Independent Lab: Inferential Statistics

In this assignment you will practice implementing statistical approaches in Python. For this assignment, please complete the following tasks using the [ect_data.txt](/data/ect_data.txt) dataset. This dataset represents an assessment of employees' adoption of a new system within an organization. Specifically, the dataset contains responses to questionnaire items about attitudes toward a system, intention to use a system, perceived usefulness of the system, and satisfaction with the system.

Linear and Multiple Regression

* Using `intent1_01` as the target variable, create a regression model using only 5 of the other numerical variables
  * Do not include `intent1_02`, `intent1_03`, and `intent1_04` as independent variables
* Assess the validity of the assumptions of
  * linearity,
  * collinearity,
  * homoscedasticity,
  * independence,
  * and normality.
* Assess your model's F-test and T-tests.
  * Which variables are significant? Based on what assessment?
  * Choose two significant variables. From a business perspective, why are they significant?
* Provide the equation for your model and interpret the results.

Save your notebook with output showing your code has been successfully executed and then submit for grading."

Thought for 14 seconds

Great—let's map out our approach and pick our five predictors before we dive into code. Here's what I suggest as our first steps:

1. **Inspect the dataset structure.**
   We need to know all of the variable names so we can choose five appropriate numerical predictors (excluding `intent1_02`, `intent1_03`, and `intent1_04`).
2. **Choose five candidate predictors.**
   Likely candidates include items like attitude toward the system, perceived usefulness, satisfaction, etc. Once we see the exact variable names we can finalize which five to include.
3. **Fit the regression model** with `intent1_01` as the outcome and our five predictors.
4. **Validate assumptions** of
   - Linearity (e.g. residuals vs. fitted plots)

- Collinearity (variance inflation factors)
- Homoscedasticity (e.g. Breusch–Pagan test)
- Independence (Durbin–Watson)
- Normality (Q–Q plot, Shapiro–Wilk)

5. **Evaluate model tests**
   - Report the overall F-test for model significance
   - Check individual t-tests for each coefficient
   - Identify which predictors are statistically significant
6. **Business interpretation**
   - Pick two significant variables, explain why they matter from an organizational-adoption perspective
   - Write out the regression equation and interpret coefficients

---

First thing: would you like me to show you the list of column names (so we can pick our five predictors)? If so, I can run a quick cell that prints them without showing full code. Let me know!

what's the print command to see the variables?

You can simply do:

```python
print(df.columns)
```

Or, if you want a plain Python list:

```python
print(df.columns.tolist())
```

First step is "Using `intent1_01` as the target variable, create a regression model using only 5 of the other numerical variables". The variables I'd like to use are t"peruse01"\t"peruse02"\t"peruse03"\t"satis03"\t"satis04"

Here's a minimal snippet to fit your OLS model with those five predictors:

python Copy Edit

```python
import statsmodels.api as sm # Define predictors and response X = df[['peruse01', 'peruse02', 'peruse03', 'satis03', 'satis04']] y = df['intent1_01'] # Add intercept X = sm.add_constant(X) # Fit model model = sm.OLS(y, X).fit() # View results print(model.summary())
```

Let me know if you'd like to proceed with this or jump straight into the assumption checks!

I revised a couple minor references, so my new cell was
"# Define predictors and response
X = ect[['peruse01', 'peruse02', 'peruse03', 'satis03', 'satis04']]
y = ect['intent1_01']

# Add intercept
X = smf.add_constant(X)

# Fit model
model = smf.OLS(y, X).fit()

View results
print(model.summary())". I got back "KeyError
Traceback (most recent call last)
Cell In[7], line 2
 1 # Define predictors and response
----> 2 X = ect[['peruse01', 'peruse02', 'peruse03', 'satis03', 'satis04']]
 3 y = ect['intent1_01']
 5 # Add intercept

File /opt/anaconda3/lib/python3.12/site-packages/pandas/core/frame.py:4108, in DataFrame.__getitem__(self, key)
 4106 if is_iterator(key):
 4107 key = list(key)
-> 4108 indexer = self.columns._get_indexer_strict(key, "columns")[1]
 4110 # take() does not accept boolean indexers
 4111 if getattr(indexer, "dtype", None) == bool:

File /opt/anaconda3/lib/python3.12/site-packages/pandas/core/indexes/base.py:6200, in Index._get_indexer_strict(self, key, axis_name)
 6197 else:
 6198 keyarr, indexer, new_indexer = self._reindex_non_unique(keyarr)
-> 6200 self._raise_if_missing(keyarr, indexer, axis_name)
 6202 keyarr = self.take(indexer)
 6203 if isinstance(key, Index):
 6204 # GH 42790 - Preserve name from an Index

File /opt/anaconda3/lib/python3.12/site-packages/pandas/core/indexes/base.py:6249, in Index._raise_if_missing(self, key, indexer, axis_name)

6247 if nmissing:
6248 if nmissing == len(indexer):
-> 6249 raise KeyError(f"None of [{key}] are in the [{axis_name}]")
6251 not_found = list(ensure_index(key)[missing_mask.nonzero()[0]].unique())
6252 raise KeyError(f"{not_found} not in index")

KeyError: "None of [Index(['peruse01', 'peruse02', 'peruse03', 'satis03', 'satis04'], dtype='object')] are in the [columns]"". Note that when I printed the variables, I had gotten " ['attitude1_01\t"attitude1_02"\t"attitude1_03"\t"attitude1_04"\t"intent1_01"\t"intent1_02"\t"intent1_03"\t"intent1_04"\t"peruse01"\t"peruse02"\t"peruse03"\t"peruse04"\t"satis01"\t"satis02"\t"satis03"\t"satis04"']"

Thought for 8 seconds

That KeyError happened because Pandas didn't split your header into separate columns – it treated the entire first line as one giant column name (you saw a single string with embedded `\t` 's when you printed `ect.columns`). To fix:

1. **Reload with the proper tab separator**, e.g.:

   ```python
   import pandas as pd ect = pd.read_csv('ect_data.txt', sep='\t', engine='python') print(ect.columns.tolist())
   ```

   Now you should see something like `['attitude1_01', 'attitude1_02', …, 'peruse01', 'peruse02', …, 'satis04']` as individual entries.

2. **Ensure you're adding the constant from the right module**. If you imported:

   ```python
   import statsmodels.api as sm
   ```

   then use:

   ```python
   X = sm.add_constant(X) model = sm.OLS(y, X).fit()
   ```

   If instead you imported:

   ```python
   import statsmodels.formula.api as smf
   ```

   note that `smf` doesn't expose `add_constant` . Either switch to `sm.add_constant` or do:

   ```python
   from statsmodels.tools import add_constant X = add_constant(X)
   ```

Once you've reloaded with `sep='\t'` and fixed the `add_constant` import, selecting `['peruse01','peruse02','peruse03','satis03','satis04']` should work without KeyErrors.

i made the changes you suggested and got "-----------

NameError Traceback (most recent call last)
Cell In[5], line 2
1 from statsmodels.tools import add_constant
----> 2 X = sm.add_constant(X)
3 model = sm.OLS(y, X).fit()
6 # Define predictors and response

NameError: name 'X' is not defined"

Thought for 5 seconds

The error is just telling you that you never defined `X` before calling `sm.add_constant(X)`. Make sure your cell runs in this order:

1. **Import** (if you haven't already):

```python
import statsmodels.api as sm
```

2. **Define** your predictors and response:

```python
X = ect[['peruse01','peruse02','peruse03','satis03','satis04']] y = ect['intent1_01']
```

3. **Add the intercept**:

```python
X = sm.add_constant(X)
```

4. **Fit** the model:

```python
model = sm.OLS(y, X).fit() print(model.summary())
```

If instead you want to use the `add_constant` you imported from `statsmodels.tools`, swap step 3 for:

```python
X = add_constant(X)
```

But the key is: **run the line that defines `X` before you try to add the constant.**

"Assess the validity of the assumptions of
* linearity,
* collinearity,
* homoscedasticity,

* independence,
* and normality."

Thought for 4 seconds

Here's how you can check each assumption—just grab these snippets and run them right after you fit your model (no need for full cells unless you want to):

---

1. Linearity

Look for a random scatter around zero with no clear pattern.

python

Copy Edit

```
# scatter plot of fitted vs. residuals import matplotlib.pyplot as plt
plt.scatter(model.fittedvalues, model.resid) plt.axhline(0,
linestyle='--') plt.xlabel('Fitted values') plt.ylabel('Residuals')
plt.show()
```

Interpretation: if you see a funnel shape or curve, the linearity assumption may be violated.

---

2. Collinearity

Compute Variance Inflation Factors (VIFs) for each predictor:

python

Copy Edit

```
from statsmodels.stats.outliers_influence import
variance_inflation_factor # assume X_const is your design matrix
including the constant vifs = pd.Series(
[variance_inflation_factor(X.values, i) for i in range(X.shape[1])],
index=X.columns ) print(vifs)
```

Rule of thumb: VIF > 5 (or definitely >10) suggests problematic multicollinearity.

---

3. Homoscedasticity

Breusch–Pagan test for constant variance of residuals:

python

Copy Edit

```
from statsmodels.stats.diagnostic import het_breuschpagan bp_stat,
bp_pvalue, _, _ = het_breuschpagan(model.resid, model.model.exog)
print('BP p-value =', bp_pvalue)
```

Interpretation: $p < 0.05 \Rightarrow$ heteroscedasticity (i.e. non-constant variance).

---

4. Independence

Durbin–Watson statistic for autocorrelation in residuals:

python

Copy Edit

```
from statsmodels.stats.stattools import durbin_watson dw =
durbin_watson(model.resid) print('Durbin–Watson =', dw)
```

Guideline: values near 2 mean residuals are uncorrelated; <1.5 or >2.5 can be concerning.

---

5. Normality

1. **Q–Q plot**

   python Copy Edit

   ```python
   import statsmodels.api as sm sm.qqplot(model.resid, line='45')
   plt.show()
   ```

2. **Shapiro–Wilk test**

   python Copy Edit

   ```python
   from scipy.stats import shapiro stat, p = shapiro(model.resid)
   print('Shapiro p-value =', p)
   ```

Interpretation: Shapiro p > 0.05 suggests residuals aren't significantly different from normal.

Once you've run these, compare the p-values and plots against the guidelines above. Let me know which tests look borderline or violated, and we can discuss remedies (e.g. transformations, adding interaction terms, switching to robust standard errors, etc.).

Uploaded image Uploaded image Uploaded image

Uploaded image Uploaded image Uploaded image

First attachment is the result for #1. Second for #2, and so on.

Here's a quick rundown of what your diagnostics tell us:

1. **Linearity**
   - **What you did:** plotted residuals vs. fitted values
   - **What you see:** a clear downward trend—as fitted values increase, residuals tend to be more negative—and clustering of points.
   - **Assessment:** that systematic pattern suggests the relationship isn't strictly linear; you may need to add polynomial terms or transform one or more

predictors.

2. **Collinearity**

   - **What you did:** computed VIFs
   - **Results:**

```cpp
const      45.58
peruse01    2.91
peruse02    2.74
peruse03    2.93
satis03     2.01
satis04     2.20
```

   - **Assessment:** all VIFs ≪ 5, so no multicollinearity concerns.

3. **Homoscedasticity**

   - **What you did:** Breusch–Pagan test → $p \approx 3.3 \times 10^{-5}$
   - **Assessment:** $p < 0.05$ rejects constant-variance; residual variance changes with fitted values (heteroscedasticity).

4. **Independence**

   - **What you did:** Durbin–Watson → 1.90
   - **Assessment:** close to 2; no serious autocorrelation in residuals.

5. **Normality**

   - **What you did:** Q–Q plot + Shapiro–Wilk → $p \approx 3.5 \times 10^{-6}$
   - **Assessment:** heavy tails and $p < 0.05$ tell us residuals deviate significantly from normal.

Bottom line:

- ✔ No collinearity issues
- ✔ Residuals appear uncorrelated (independence is fine)
- ⚠ Linearity, homoscedasticity, and normality assumptions are violated

Next steps to consider:

- Transform predictors or outcome (e.g., log or square-root) or add non-linear terms
- Fit a weighted least-squares or use robust (HC) standard errors to address heteroscedasticity
- Check if outliers are driving non-normality, or consider a generalized linear model if normality can't be recovered

Let me know which direction you'd like to explore first!

what are VIFs?

Variance Inflation Factors (VIFs) quantify how much the variance (i.e. the squared standard error) of an estimated regression coefficient is "inflated" because of linear relationships among the predictors. In other words, they're a measure of **multicollinearity**.

- **How it's computed**
  For each predictor X_j, you regress it on all the *other* predictors and compute the R_j^2 of that auxiliary regression. Then

$$\mathrm{VIF}_j = \frac{1}{1 - R_j^2}.$$

- **Interpretation**
  - $\mathrm{VIF} = 1 \Rightarrow R_j^2 = 0$, so X_j is uncorrelated with the others.
  - $\mathrm{VIF} > 1 \Rightarrow$ some collinearity.
  - Common rules of thumb:
    - VIF > 5 ⇒ moderate collinearity worth watching.
    - VIF > 10 ⇒ serious collinearity; coefficient estimates may be unstable.
- **Why it matters**
  High multicollinearity inflates the standard errors of your coefficients, making it harder to detect which predictors are truly significant (even if they have a real effect).

Since all of your VIFs were between about 2.0 and 3.0, you don't have a collinearity problem here.

but the shapiro value was Shapiro p-value = 3.546790585127104e-06 ?

The Shapiro–Wilk test and VIFs are checking completely different assumptions:

1. **VIFs** look at *multicollinearity* among your predictors (i.e. how much they overlap with each other).
2. The **Shapiro–Wilk test** looks at *normality of your residuals*.

- Your VIFs (all ≪5) told you there's no worrisome collinearity.
- Your **Shapiro p-value ≈3.5 × 10^{-6}** (≪0.05) tells you that the residuals significantly deviate from a Normal distribution.

So yes—the low Shapiro p-value flags a violation of the **normality** assumption, but it has nothing to do with collinearity. It means one or more of:

- Heavy-tailed or skewed residuals
- Potential outliers
- Model misspecification

Next steps might be:

- Identify and examine outliers
- Transform the outcome (e.g. log, square-root) or a predictor
- Try a robust regression (e.g. Huber or White's robust SEs)
- Consider a different link/function if you can't recover normality

Let me know which of those you'd like to tackle first!

"Assess your model's F-test and T-tests. " How do we do that?

You get both the overall F-test and the individual t-tests right out of your fitted model object. Here's how:

```python
# Overall model F-test print("F-statistic:", model.fvalue) print("F-test p-value:", model.f_pvalue) # Individual coefficient t-tests print("Coefficients:\n", model.params) print(" t-values:\n", model.tvalues) print(" p-values:\n", model.pvalues)
```

Interpretation

1. **F-test**
   - **Null hypothesis:** All slope coefficients = 0 (i.e. none of your five predictors add explanatory power)
   - If the F-statistic is large and its p-value < your α (e.g. 0.05), you reject that null and conclude the model as a whole is statistically significant.
2. **t-tests (one per predictor)**
   - **Null hypothesis for each:** That predictor's coefficient = 0
   - Look at each coefficient's p-value:
     - p < 0.05 ⇒ that predictor has a statistically significant relationship with `intent1_01` (keeping the others constant).
     - p ≥ 0.05 ⇒ no evidence it contributes beyond the other variables.

Once you've printed those out, identify which predictors have p < 0.05. Those are your "significant" variables to discuss for the business interpretation step.

"bF-statistic: 35.0244209799111
F-test p-value: 1.789524331981252e-25
Coefficients:
const 1.613962
peruse01 0.275291
peruse02 0.029308
peruse03 0.190271
satis03 0.128428
satis04 0.181164
dtype: float64
t-values:
const 4.520465
peruse01 3.277190
peruse02 0.408137
peruse03 2.419460
satis03 2.070689
satis04 2.519005
dtype: float64
p-values:

const 0.000011
peruse01 0.001241
peruse02 0.683621
peruse03 0.016461
satis03 0.039705
satis04 0.012572
dtype: float64" * Which variables are significant? Based on what assessment?
* Choose two significant variables. From a business perspective, why are they significant?

Which variables are significant?

Based on the individual t-tests (p-values < 0.05), four predictors are significant:

- **peruse01** ($p \approx 0.0012$)
- **peruse03** ($p \approx 0.0165$)
- **satis03** ($p \approx 0.0397$)
- **satis04** ($p \approx 0.0126$)

(peruse02 is not significant, $p \approx 0.68$.)

The overall F-test ($F \approx 35.0$, $p \approx 1.8 \times 10^{-25}$) confirms the model as a whole explains a significant amount of variance in intent1_01.

Two business-driven interpretations

1. **peruse01 ($\beta \approx 0.275$, $p \approx 0.0012$)**
   - *What it measures:* Users' belief that "this system improves my job performance" (e.g. perceived usefulness).
   - *Why it matters:* When employees see a clear productivity boost, they're far more likely to intend to use the system. Investing in features or training that highlight performance gains will therefore drive adoption.
2. **satis04 ($\beta \approx 0.181$, $p \approx 0.0126$)**
   - *What it measures:* Overall satisfaction with the system (e.g. "I'm pleased with how the system works").
   - *Why it matters:* Even if a tool is useful, low satisfaction (due to usability issues, support gaps, etc.) erodes intention to use. Focusing on user experience improvements and responsive support can therefore boost long-term adoption.

These two variables explain both the **"can-do"** belief (usefulness) and the **"want-to"** feeling (satisfaction), which together form the core of driving user intent.

* Provide the equation for your model and interpret the results.

The fitted regression equation is:

$$\widehat{\text{intent1\_01}} = 1.61396 + 0.27529\,\text{peruse01} + 0.02931\,\text{peruse02} + 0.19027\,\text{peruse03} + 0.12843\,\text{satis03} + 0.18116\,\text{satis04}$$

Interpretation of the coefficients

- **Intercept (1.614):**
  When all five predictors are zero (theoretical baseline), the model predicts intent ≈ 1.61.
- **peruse01 (0.275, p ≈ 0.001):**
  A one-point increase in "perceived usefulness item 1" boosts predicted intent by 0.275, holding the others constant. This is the largest and most significant driver.
- **peruse02 (0.029, p ≈ 0.68):**
  Not statistically significant—little evidence this specific usefulness item adds beyond the others.
- **peruse03 (0.190, p ≈ 0.016):**
  A one-point rise in "perceived usefulness item 3" raises intent by 0.19. Significant at the 5 % level, so this aspect of usefulness meaningfully influences intent.
- **satis03 (0.128, p ≈ 0.040):**
  Each one-point gain in "satisfaction item 3" increases predicted intent by 0.128. Significant, indicating that this facet of satisfaction matters for adoption.
- **satis04 (0.181, p ≈ 0.013):**
  A one-point uptick in "satisfaction item 4" is associated with a 0.181 increase in intent. Also significant, underscoring the role of overall satisfaction.

Bottom-line:
Employees' **perceptions of usefulness** (especially peruse01 and peruse03) and their **satisfaction** (satis03 and satis04) all have positive, statistically significant associations with their stated intention to use the system. Peruse01 is the single strongest predictor in this model.

give me the equation is a format i can copy paste

Here's the equation ready to copy-paste:

text Copy Edit

```
intent1_01_pred = 1.613962 + 0.275291*peruse01 + 0.029308*peruse02 +
0.190271*peruse03 + 0.128428*satis03 + 0.181164*satis04
```